

Advant[®] OCS
with Master software

MultiVendor Interface
Advant[®] Controller 400 Series

User's Guide

3BSE 000 531R0101 Rev A



Use of **DANGER**, **WARNING**, **CAUTION**, and **NOTE**

This publication includes, **DANGER**, **WARNING**, **CAUTION**, and **NOTE** information where appropriate to point out safety related or other important information.

DANGER	Hazards which could result in severe personal injury or death
WARNING	Hazards which could result in personal injury
CAUTION	Hazards which could result in equipment or property damage
NOTE	Alerts user to pertinent facts and conditions

Although **DANGER** and **WARNING** hazards are related to personal injury, and **CAUTION** hazards are associated with equipment or property damage, it should be understood that operation of damaged equipment could, under certain operational conditions, result in degraded process performance leading to personal injury or death. Therefore, comply fully with all **DANGER**, **WARNING**, and **CAUTION** notices.

TRADEMARKS

Advant is registered trademark of ABB Asea Brown Boveri Ltd, Switzerland.

ABB Master, MasterPiece, MasterBus and MasterNet are registered trademarks of ABB Asea Brown Boveri Ltd, Switzerland.

Advant Controller is a trademark of ABB Asea Brown Boveri Ltd, Switzerland.

HP and all HP-products mentioned in this publication are registered trademarks of Hewlett Packard Corporation.

Westermo is a registered trademark of Westermo Teleindustri AB, Sweden.

NOTICE

The information in this document is subject to change without notice and should not be construed as a commitment by ABB Industrial Systems AB. ABB Industrial Systems AB assumes no responsibility for any errors that may appear in this document.

In no event shall ABB Industrial Systems AB be liable for direct, indirect, special, incidental or consequential damages of any nature or kind arising from the use of this document, nor shall ABB Industrial Systems AB be liable for incidental or consequential damages arising from use of any software or hardware described in this document.

This document and parts thereof must not be reproduced or copied without ABB Industrial Systems AB's written permission, and the contents thereof must not be imparted to a third party nor be used for any unauthorized purpose.

The software described in this document is furnished under a license and may be used, copied, or disclosed only in accordance with the terms of such license.

CE MARKING

This product meets the requirements specified in EMC Directive 89/336/EEC and in Low Voltage Directive 72/23/EEC.

Copyright © ABB Industrial Systems AB 1998.

TABLE OF CONTENTS

Chapter 1 - Introduction

1.1	General Information.....	1-1
1.2	Equipment Requirements.....	1-2
1.2.1	Target System.....	1-2
1.2.2	Development System	1-3
1.3	Manual Organization	1-4
1.4	Conventions	1-5
1.5	Related Documentation.....	1-6
1.6	Release History	1-6
1.7	Terminology.....	1-7
1.8	Product Overview	1-7

Chapter 2 - Installation

2.1	Site Planning Environment	2-1
2.2	Setup	2-1
2.2.1	Hardware.....	2-1
2.2.2	Software	2-9
2.2.2.1	MVIBase and User Development Structure.....	2-9
2.2.2.2	SDE	2-11
2.2.2.3	SLIP	2-11
2.3	Shut-down Procedures	2-11
2.4	Start-up Procedures.....	2-11
2.5	Product Verification	2-12

Chapter 3 - Configuration/Application Building

3.1	Design Consideration.....	3-1
3.2	Capacity & Performance.....	3-2
3.3	MVI Development Environment Setup	3-2
3.3.1	SDE	3-2
3.3.2	MVIBase.....	3-3
3.3.3	Host/Target Internet Addresses.....	3-3
3.3.3.1	Configuring the Host System	3-3
3.3.3.2	Configuring the Target System	3-5
3.3.4	SLIP Interface	3-5
3.3.5	Test Directory	3-8
3.3.6	Download Script	3-8
3.3.7	Debugger Initialization	3-9
3.3.8	Preparation for System Building.....	3-9

CONTENTS (continued)

3.4	Tutorial	3-11
3.5	Application Procedures	3-14
3.5.1	Protocol Handler Development.....	3-14
3.5.1.1	Create Source Code	3-17
3.5.1.2	Compile the Source Code	3-17
3.5.1.3	Download to Target	3-19
3.5.1.4	Debug in Target	3-19
3.5.1.5	Build a Prommed System	3-20
3.5.2	Demo Protocol Application Building for Advant Controller 400.....	3-22
3.5.2.1	Master Functions	3-24
3.5.2.2	Slave Functions	3-24
3.6	Configuration/Application Building Menus	3-25
3.6.1	MVI UART Interface	3-25
3.6.1.1	Read Descriptor and Write Descriptor Data Structures.....	3-25
3.6.1.2	UART Interrupt reasons	3-27
3.6.1.3	Open	3-28
3.6.1.4	Close.....	3-29
3.6.1.5	Read.....	3-29
3.6.1.6	Write	3-30
3.6.1.7	Ioctl.....	3-30
3.6.1.8	Select	3-33
3.6.2	MVI Watchdog Timers.....	3-35
3.6.2.1	Open	3-35
3.6.2.2	Close.....	3-36
3.6.2.3	Ioctl.....	3-36
3.6.2.4	Select	3-37
3.6.3	MVI Checksum Library	3-39
3.6.3.1	MviCrc_16.....	3-39
3.6.3.2	MviBcc	3-43
3.6.3.3	MviLrc	3-44
3.6.3.4	MviCrc_8.....	3-45
3.6.3.5	MviSum	3-45
3.6.3.6	MviSkipControl.....	3-46
3.6.3.7	MviAddControl	3-47

CONTENTS (continued)

3.6.4	MVI Runtime Library	3-47
3.6.4.1	MviGetNodeDLEntry	3-48
3.6.4.2	MviGetNodeStatusInfo	3-49
3.6.4.3	MviGetNodeDiagInfo	3-49
3.6.4.4	MviInitUART	3-50
3.6.4.5	MviCreateUARTDevice.....	3-51
3.6.4.6	MviOpenUARTDevice	3-51
3.6.4.7	MviReqData	3-51
3.6.4.8	MviGetData.....	3-52
3.6.4.9	MviSendData	3-52
3.6.4.10	MviSelect	3-53
3.6.4.11	MviSendLinkStatus.....	3-55
3.6.4.12	MviCreatePHIDevices	3-55
3.6.4.13	MviCreateDLIDevices	3-56
3.6.4.14	MviOpenPHIDevices	3-56
3.6.4.15	MviOpenDLIDevices.....	3-57
3.6.4.16	MviCreateTimerDevices	3-57
3.6.4.17	MviOpenTimerDevices.....	3-58
3.6.4.18	MviFetchSetupInfo	3-59
3.6.4.19	MviInitDLStruct.....	3-59
3.6.4.20	MviInitGlobalDataStruct	3-60
3.6.4.21	MviMakeWDTDevName.....	3-60
3.6.4.22	MviPHICleanup	3-61
3.6.4.23	MviDLICleanup	3-61
3.6.4.24	MviPHIWaitForModemSignals	3-62
3.6.4.25	MviDLIWaitForModemSignals	3-63
3.6.4.26	MviSysMess.....	3-64
3.6.5	MVI Protocol Handler Interface	3-65
3.6.5.1	MviReqPHISetup	3-70
3.6.5.2	MviGetPHIReqSetup	3-71
3.6.5.3	MviSendPHISetupInfo	3-71
3.6.5.4	MviGetPHISetupInfo	3-72
3.6.5.5	MviSendPHIWatchdog	3-72
3.6.5.6	MviGetPHIWatchdog.....	3-73
3.6.5.7	MviSendPHIDiagInfo	3-73
3.6.5.8	MviGetPHIDiagInfo	3-74
3.6.5.9	MviSendPHIStatusInfo	3-74
3.6.5.10	MviGetPHIStatusInfo	3-75

CONTENTS (continued)

3.6.5.11	MviSendPHILocalDownCmnd	3-75
3.6.5.12	MviGetPHILocalDownCmnd	3-76
3.6.5.13	MviSendPHILocalDownResp	3-76
3.6.5.14	MviGetPHILocalDownResp	3-77
3.6.5.15	MviSendPHIDownCmnd	3-77
3.6.5.16	MviGetPHIDownCmnd.....	3-78
3.6.5.17	MviSendPHIDownResp	3-78
3.6.5.18	MviGetPHIDownResp	3-79
3.6.5.19	MviSendPHIUpCmnd	3-79
3.6.5.20	MviGetPHIUpCmnd	3-80
3.6.5.21	MviSendPHIUpResp	3-80
3.6.5.22	MviGetPHIUpResp	3-81
3.6.6	MVI DataLink Interface.....	3-81
3.6.6.1	MviSendDLIDownCmnd	3-83
3.6.6.2	MviGetDLIDownCmnd	3-84
3.6.6.3	MviSendDLIDownResp	3-84
3.6.6.4	MviGetDLIDownResp	3-85
3.6.6.5	MviSendDLIUpCmnd	3-85
3.6.6.6	MviGetDLIUpCmnd	3-86
3.6.6.7	MviSendDLIUpResp	3-86
3.6.6.8	MviGetDLIUpResp	3-87
3.6.7	Protocol Configuration and Application Building for Advant Controller 400.....	3-88
3.6.7.1	MS for MVI Protocol Configuration.....	3-88
3.6.7.2	Command MS used in Master Mode.....	3-116
3.6.7.3	Data MS for Data Transfer	3-125
3.6.7.4	Type Circuits for MVI Protocol Data Flow Control	3-133
3.6.8	Host System Interface in Advant Controller 400	3-136
3.6.8.1	Network Addresses.....	3-136
3.6.8.2	BufAllocate	3-137
3.6.8.3	BufRelease.....	3-138
3.6.8.4	TransceiverInit.....	3-138
3.6.8.5	TranHostInfoGet.....	3-139
3.6.8.6	TranChanDevCreate	3-139
3.6.8.7	Open	3-140
3.6.8.8	Read	3-141
3.6.8.9	Write	3-141
3.6.8.10	Close.....	3-142

CONTENTS (continued)

Chapter 4 - Runtime Operation

4.1	Operating Overview.....	4-1
4.2	Runtime Tutorial.....	4-1
4.3	Operating Instructions.....	4-1

Chapter 5 - Maintenance

5.1	Preventative Maintenance.....	5-1
5.2	Hardware Indicators.....	5-1
5.3	Error Messages	5-2
5.4	Fault Finding and User Repair.....	5-8
5.4.1	SLIP Communication	5-8
5.4.2	Cross-Development Tools.....	5-9
5.4.2.1	Editor.....	5-9
5.4.2.2	Compiler.....	5-9
5.4.2.3	Debugger	5-9
5.4.3	MVI Demo Protocol and other Asyn. Communication Protocols	5-10

Appendix A - Code Example

A.1	Example: Protocol Handler.....	A-1
-----	--------------------------------	-----

Appendix B - MVI Demo Protocol Configuration

B.1	MVI Demo Protocol Configuration and Application Building for Advant Controller 400.....	B-1
B.1.1	MS for MVI Demo Protocol Configuration	B-1
B.1.1.1	Line Characteristics MS	B-1
B.1.1.2	Network Configuration MS	B-6
B.1.1.3	RTU Status MS	B-9
B.1.1.4	Register Address MS	B-17
B.1.2	Command MS used in Master Mode	B-20
B.1.2.1	Control Commands	B-20
B.1.2.2	Read Commands	B-23
B.1.2.3	Write Commands.....	B-25
B.1.2.4	Data MS for Data Transfer.....	B-27

Appendix C - A Makefile Example

C.1	Makefile	C-1
-----	----------------	-----

Appendix D - Hardware Module

D.1	CI535 Communication Interface	D-1
D.2	CI536 Communication Interface	D-3

Chapter 1 Introduction

1.1 General Information

MultiVendor Interface (MVI) is an environment you can use to interface external vendors' PLCs and RTUs to Process Stations in the Advant Controller 400 Series.

MVI consists of a Development Environment, a Runtime Environment and a Demo Protocol Handler.

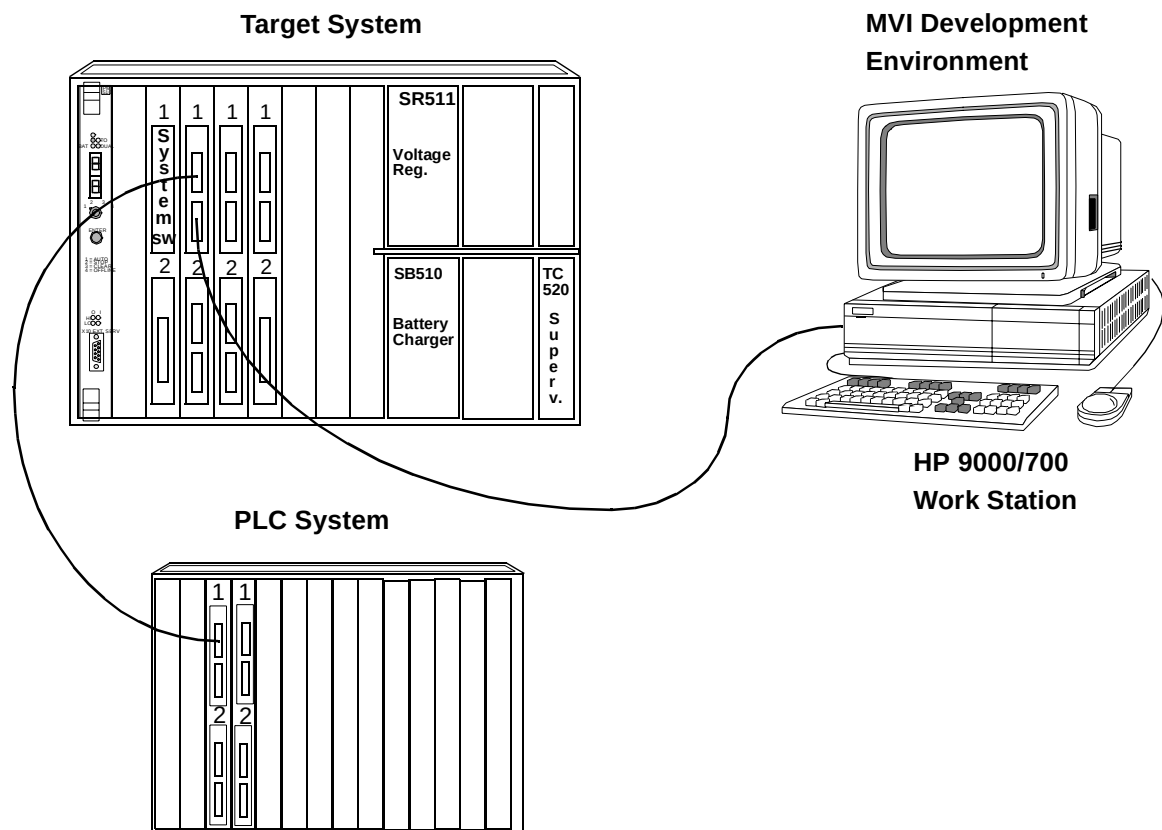


Figure 1-1. MultiVendor Interface

The MVI Development Environment is the software base for the development of alien RTU/PLC protocols. You can use protocol developed in MVI in the above mentioned series of Advant Controllers. The MVI Development Environment runs on an HP 9000/700 work station.

The MVI Runtime Environment is the environment in the target system where the Protocol Handler developed executes.

The Demo Protocol Handler serves as an example of how to implement a Protocol Handler using MVI.

This manual describes MVI. It provides step-by-step instructions for installation, configuration and development of new protocols using MVI. It is targeted towards engineers creating alien protocols for Advant Controllers within the ABB DCS.

Knowledge of configuration and application programming of the Advant Controller 400 Series, as well as C programming and general communication protocol knowledge, is required. Knowledge of configuration and application programming of some of the MVI standard protocols is recommended.

1.2 Equipment Requirements

1.2.1 Target System

Communication Hardware in the Advant Controller 410

- CI535 Communication submodule, equipped with 512 KB RAM and 512 KB Flash PROM, for connection of two communication links. The CI535 submodule is typically used for production of the protocols developed, or during development of small protocols, where the smaller memory is sufficient to accommodate the protocol together with the extra software support for communication with the development system, remote debugging, and so forth.

The communication submodule is connected to one of the four submodule positions on the main processor module PM150.

Communication Hardware in the Advant Controller 450

- CI535 Communication submodule, equipped with 512 KB RAM and 512 KB Flash PROM, for connection of two communication links.
- A free submodule position for the CI535 submodule on one of the following submodule carriers:
 - SC510 Carrier without local processor (one submodule position can also be used for other functions, that is, MB 300).
 - SC520 Carrier with local processor for MB 300 (one submodule position can also be used for other functions, that is, MB 300).

Modems and Cables

The communication submodule CI535 is equipped with two independent, CCITT V.24 (RS232-C), asynchronous communication ports. For connection of the serial link, the following modems and cables are recommended:

- Westermo LA-01¹
Full duplex modem for multidrop communication up to 10 km.
- TC562
Full duplex modem for cable length up to 10 km. Point-to-point up to 1 km at 19200 bps. Power 24 V DC.
- TK577
TK577 is a 3 m cable with a nine-position D-Sub Connector Receptacle (female) for connection to one port on the front of the CI535 submodule and a 25-position D-Sub Connector Plug (male) for connection to the modem LA-01 or Uniline 2.4. (For more information on hardware setup, see [Section 2.2.1, Hardware.](#))
- TK595
Cable with a nine-position D-Sub Connector Receptacle (female) for connection to one port on the front of the CI535 submodule and a 9-position D-Sub Connector Plug (male) for connection to the modem TC562.
- Bus cable with two twisted pairs, each pair individually shielded, minimum conductor area 0.22 mm². You can use the following cable or a similar cable:
 - Belden Type 9406
 - Kabelmetal Type DUE 4552
 - Alpha Type 6032
 - Norsk Kabel Type O6802.

1.2.2 Development System

Work Station

- HP 9000/700 work station with one free RS232-C serial communication port.
- DAT driver for installation of software.

Software

- SDE (optional)
- HP-UX 9.0 or later
- MVIBase *2.0.

1. Ordered from Westermo Teleindustri AB, St. Sundby, Sweden.

Communication Hardware in the Advant Controller 400 Series

- CI536 Communication submodule, equipped with 1 MB RAM and 1 MB Flash PROM, for connection of two communication links. The CI536 submodule is typically used during development of larger, more complex protocols, where the larger memory is needed.

NOTE

You can use the CI536 Communication submodule for development purposes only. It is not part of the Advant Controller 400 Series.

PROM Programmer

- A PROM programmer which supports programming of 28F020 type Flash PROMs.

Engineering Tools

To build the data base and the PC program in the Advant Controller 400, you need an:

- Engineering Station.

1.3 Manual Organization

Figure 1-2 shows the basic structure of the Advant System’s documentation. Each document, describing either hardware or software, is built around this structure to make it easy for you to locate related information in any of the documents.

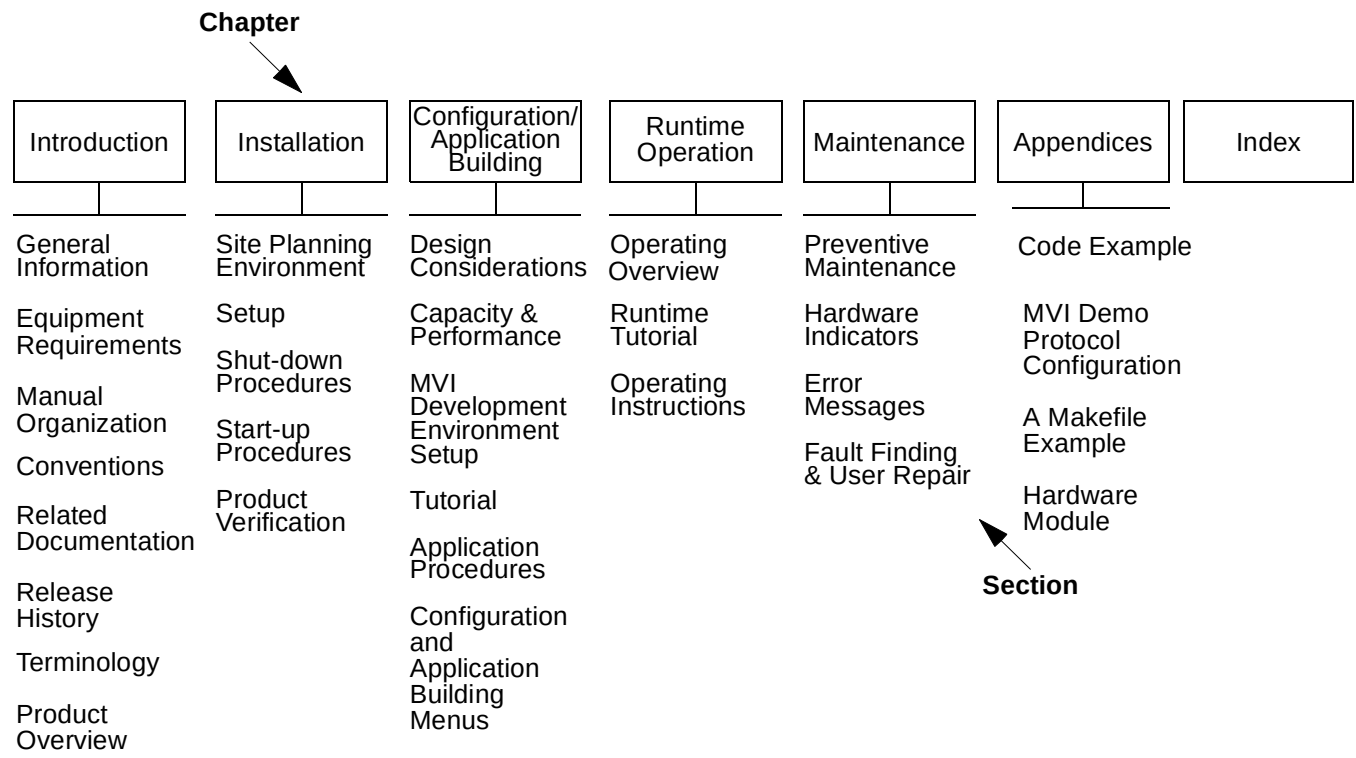


Figure 1-2. Manual Organization Diagram

Introduction

[Chapter 1](#), serves as an introduction to the MVI Development Environment.

Installation

[Chapter 2](#), tells you how to set up the communication hardware in AC400. You can also find descriptions of how to install the software, the shut-down and start-up procedures, and how to verify correct operation after power-up.

Configuration and Application Building

[Chapter 3](#), gives you a description of how to configure and use the MVI Development Environment. The chapter includes a tutorial which describes the development of a typical Protocol Handler. In [Section 3.5, Application Procedures](#), you are given brief guidelines on how to break your application down into appropriate modules along with a step-by-step description of the different stages in the development of a communication Protocol Handler. How to build an application for the MVI Demo Protocol is also described. The remainder of the chapter is reference material.

Runtime Operation

[Chapter 4](#), provides information on how to operate some of the MVI facilities.

Maintenance

[Chapter 5](#), provides help and documentation relative to errors that may occur when you are using the MVI Development Environment.

1.4 Conventions

All input values to data base elements which require you to enter the value are **boldfaced**.

All commands described, which are given on the command line in the development system or in the target system, are **boldfaced**.

All references are in *italics*.

All program code examples are presented in the font shown below:

```
main()  
{  
    /* This is a sample program */  
    printf("Hello world!\n");  
}
```

\$ Prompt in Korn shell (HP-UX command interpreter).

-> Prompt in VxWorks shell.

(vxgdb) Prompt in VxGDB command interpreter.

1.5 Related Documentation

For further information about configuration and application programming, refer to the following documents.

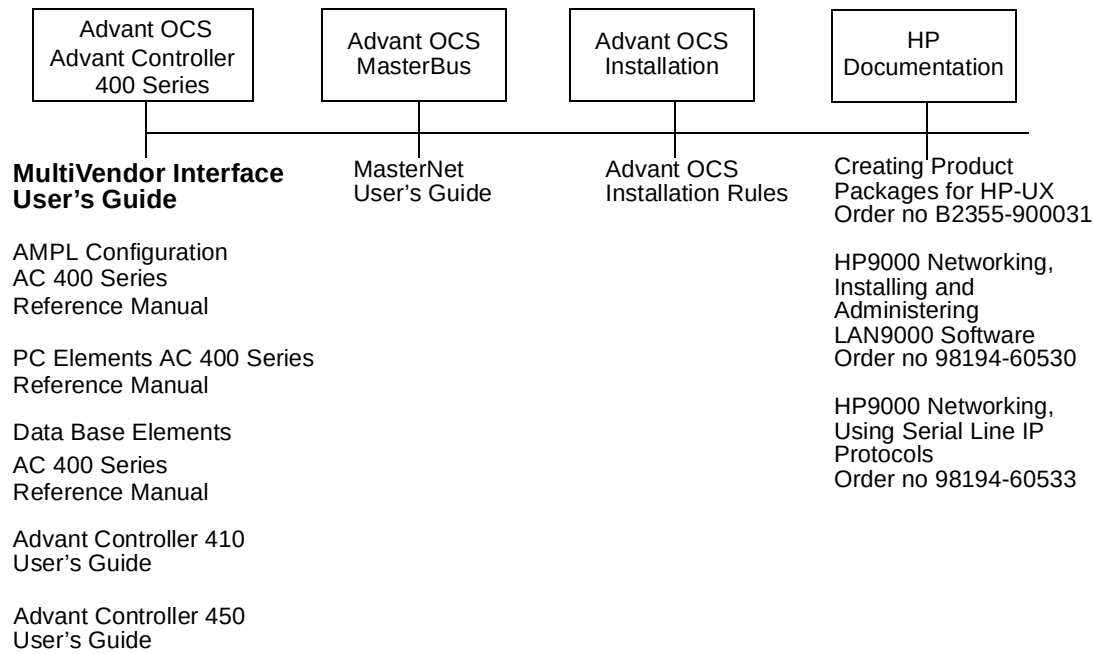


Figure 1-3. Related Documentation Tree

1.6 Release History

This is the initial release of the MultiVendor Interface for the Advant Controller 410 and Advant Controller 450.

1.7 Terminology

Abbreviations, acronyms and terms that you should be familiar with:

- **AC 400** is an abbreviation for the Advant Controller 400 Series (Advant Controller 410 and Advant Controller 450).
- **AMPL** is an acronym for the **ABB Master Programming Language**. AMPL is a function-block language with graphic presentation that is specifically oriented towards process control. The AMPL is used for configuration and application building in Advant Controllers.
- **Controller** refers to Advant Controllers.
- **Command MS** is a normal MS (see below) defining the command used to control slaves.
- **Configuration MS** is a normal MS used to configure the CI535 submodule for the communication.
- **DAT** is a concept in the data base that holds a data value (one Real (4 bytes), one Integer (2 or 4 bytes) or 32 Boolean values). The data values to MS are held by connected DAT elements.
- **Data MS** is the MS used for the actual data transfer.
- **DS** is an abbreviation for **Data Set**, which is a data base concept in Advant OCS. It is used to transfer blocks of data in an Advant OCS network.
- **MB 300 or MasterBus 300** is a high-speed bus used to interconnect Controllers, Operator Stations and Communication Stations.
- **MP** is an abbreviation for the ABB MasterPiece.
- **MS** is a special type of Data Set used by the MultiVendor Interface (**MultiVendor Interface Data Set**).
- **MVI** is an abbreviation for the Advant OCS MultiVendor Interface.
- **PC program** (Process Control program) is a program written in AMPL.
- **RTU** is an abbreviation for Remote Terminal Unit.
- **SDE Software Development Environment** is an environment for project management developed by ABB.

1.8 Product Overview

The MultiVendor Interface Environment consists of the following base components:

- MVI Development Environment.
- MVI Runtime Environment.
- MVI Demo Protocol.

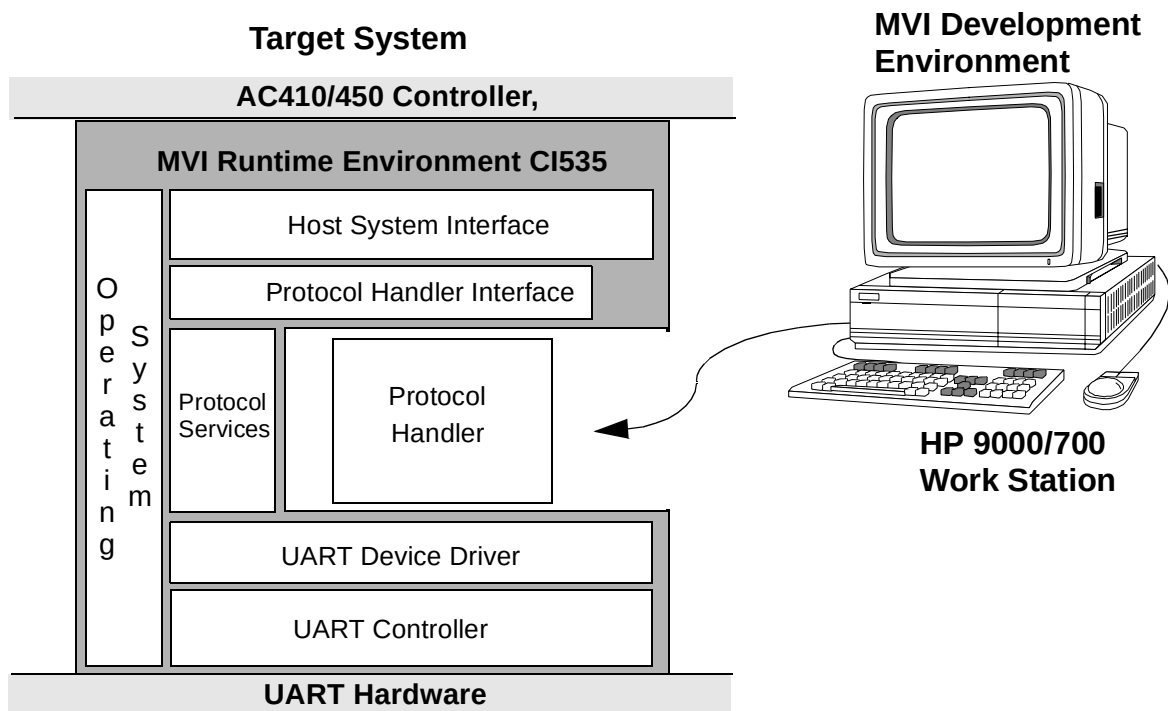


Figure 1-4. MultiVendor Interface

MVI Development Environment

The MVI Development Environment is the software base for the development of RTU/PLC protocols. It consists of SDE for project management; a set of tools for building the protocol developed; and a set of header files which define the MVI runtime types, service functions and devices. The tools include an editor, compiler, linker and remote debugger. The MVI Development Environment is running on an HP 9000/700 series work station.

A serial port on the work station is used for communication with the target system during development. The protocol used is SLIP (Serial Line Internet Protocol), which is a protocol for running TCP/IP applications like rlogin, telnet, etc., over a serial line. The MVI Development Environment uses rlogin to log on to the target system to download software, control the target system via the local command interpreter (shell), etc. The serial line is also used by the remote debugger, which sends debug commands and receives debug information on the serial line.

HP 9000 computers implement SLIP (and other serial line IP protocols) with a feature called Point-to-Point Link (PPL). The PPL facility is included in HP's LAN9000 product supplied with HP-UX 8.0 or later versions. PPL includes a command program (named *ppl*). You can use PPL to set up:

- Dial-in to HP 9000 serial IP lines.
- Dial-out from HP 9000 serial IP lines.
- Direct connection to HP 9000 serial IP lines.

The MVI Development Environment uses only **direct connection** between the HP 9000/700 work station and the CI535 communication submodule.

MVI Runtime Environment

The MVI Runtime Environment is the environment in which the protocol developed is running.

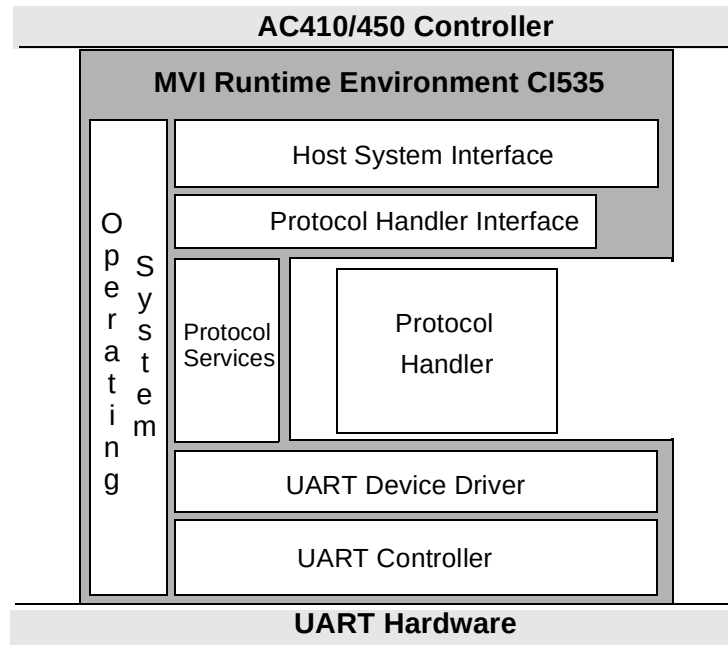


Figure 1-5. MVI Runtime Environment

Besides the operating system, the MVI Runtime Environment consists of a number of library modules comprising the interfaces and support functions used by the Protocol Handler.

- MVI UART Interface, consisting of the UART device driver and the UART controller, is the interface towards the UART hardware. It provides functions for:
 - Initializing software and UART hardware.
 - Activating/deactivating trigger actions. The UART can be set to trigger on different trigger conditions such as a certain number of characters, one or several (up to five) trigger characters with an and/or condition between them, or a combination of the above. You can also specify whether you want the trigger character(s) and/or the characters that come before the trigger character(s) to be included in the message or ignored.
 - Writing and reading messages to/from the serial link.
 - Waiting for an event from the UART to occur. The mechanism for waiting on events from one or several of the devices supported enables a Protocol Handler to suspend itself on the UART and possibly other devices such as a Watchdog Timer or a FIFO. The execution is then resumed when an event occurs on one or several of the devices on which it has been waiting.

- MVI Protocol Services is a set of support functions for the Protocol Handler. They provide functionality for:
 - Creating, opening, initializing and closing devices.
 - Read/write from/to devices, that is, the UART.
 - Allocating, initializing and deallocating data structures.
 - Initializing interfaces, that is, MVI Protocol Handler Interface.
 - Accessing certain parts of data structures.
 - Task suspension on one or several devices, waiting for events to occur.
 - Establishing and taking down connections between the host system and a node on the serial link.
 - Calculating check sums using a number of different algorithms.
- MVI Protocol Handler Interface is a set of functions for communication with the host system.

MVI Demo Protocol

The MVI Demo Protocol included in MVI is a simple protocol provided as an example of how a Protocol Handler can be implemented.

Chapter 2 Installation

2.1 Site Planning Environment

See the *Advant OCS Installation Rules* manual for guidelines on earth connection, grounding, power supply and so forth.

The following requirements apply for the Development system:

- HP 9000/700 with HP-UX version 9.0 or later.
- SDE installed (optional).

The following requirements apply for the Advant Controller system:

- Advant Controller 410, with a CI535 communication submodule or
- Advant Controller 450, with a SC510 or SC520 carrier and a CI535 communication submodule.

2.2 Setup

2.2.1 Hardware

The following hardware must be available for the development:

- Advant Controller 410 or,
- Advant Controller 450, with a SC510 or SC520 carrier.
- CI535 communication submodule.
- HP 9000/700 series workstation with one free serial port.
- A PROM Programmer capable of programming 28F020 type Flash PROMs.

The CI535 submodule is equipped with two independent, CCITT V.24 (RS232-C), asynchronous communication ports, see [Figure 2-1](#).

- In Advant Controller 410, the CI535 submodule is connected to one of the four submodule positions on the main processor module PM150, see [Figure 2-3](#).
- In Advant Controller 450, the CI535 submodule is connected to a free submodule position on one of the following submodule carriers: SC510 or SC520. [Figure 2-1](#) shows two CI535 submodules connected to a SC510 carrier.

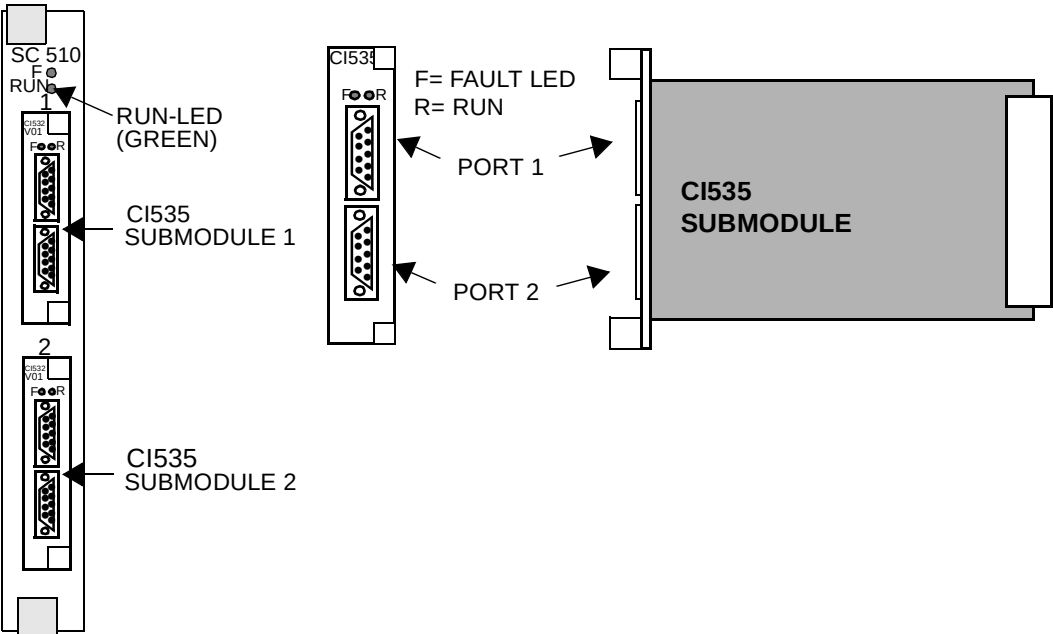


Figure 2-1. SC510 Carrier and CI535 Submodule

The setup for each port on the CI535 submodule is accomplished with the data base element CI535¹ in Advant Controller 400 Series, see Figure 2-2. There are no strappings on the CI535 submodule and no strappings on the carrier boards SC510 and SC520.

Record number		MVI/MVIn	
		MVI Free Prog	(344.n)
		Base part	
	MVIn	1	NAME
	0	3	BUS
	0	4	STATION
(AC 450/AC 410)	3-8/1	5	POSITION
(AC 450/AC 410)	1-2/1-4	6	SUBPOS
	1	10	IMPL
	1	11	SERVICE
CI535		7	TYPE
(Own node number)	1-99	18	NODE
	OFF	28	CONSOLE
		Ch1 Comm. Param.	
	1-9	22	NET1
	0	25	SET_NET1
			VALID1
			DSR1
			CTS1
			DCD1
			RI1
		Ch2 Comm. Param.	
	1-9	23	NET2
	0	26	SET_NET2
			VALID2
			DSR2
			CTS2
			DCD2
			RI2

Figure 2-2. Data Base Element CI535 for MVI Communication Submodule CI535 in Advant Controller 400

You must check or fill in the correct values on the CI535 data base element (element type is MVI and call name is CI535). For further description, please refer to the *Data Base Elements Advant Controller 400 User's Guide*.

1. The call name for the data base element is CI535 and the element type is MVI.

Table 2-1. Properties for the Data Base Element CI535 in Advant Controller 400

Terminal	Value in AC 450	Value in AC 410	Description
CI535 Record number	1 - 5	1 - 2	CI535 submodule number. Record number for the CI535 data base element for the communication port defines the submodule number. Record no 1 => submodule = 7 Record no 2 => submodule = 8 Record no 3 => submodule = 9 (only valid for AC 450) Record no 4 => submodule = 10 (only valid for AC 450) Record no 5 => submodule = 11 (only valid for AC 450)
1 NAME			Unique NAME of the element.
3 BUS	0	0	Not used
4 STATION	0	0	Not used
5 POSITION	2 - 8	1	POSITION of the carrier board in the CPU rack, see figure below. To change the POSITION in operation mode, the IMPL must be set to 0. See figure below.
6 SUBPOS	1 - 2	1 - 4	SUBPOSITION on the carrier board, see figure below. To change the SUBPOS in operation mode, the IMPL must be set to 0. See figure below.
10 IMPL	1	1	IMPLemented =1 to set CI535 submodule in normal operation
11 SERVICE	1	1	SERVICE = 1 to set CI535 submodule in normal operation SERVICE = 0 to stop the communication on the communication bus
7 TYPE	CI535	CI535	Always CI535
18 NODE	1 - 99	1 - 99	Set to AC 400's own node number. If node number of the AC 400 is already defined, NODE can only be set to that node number. In order to modify a node number >0, a COLD START of the Advant Controller 400 is necessary.
28 CONSOLE	OFF	OFF	CONSOLE = ON to enable console function on channel 2. Used for test purposes.
16 WARNING	0 - 1	0 - 1	Warning flag
17 ERR	0 - 1	0 - 1	Error flag
9 ERRTYPE	0 - n	0 - n	Error type. See Section 5.3, Error Messages , for translation.
20 PROTOCOL			Protocol = 'protocol' (set by the system according to the set-up of the Line Characteristics MS)

Table 2-1. Properties for the Data Base Element CI535 in Advant Controller 400 (Continued)

Terminal	Value in AC 450	Value in AC 410	Description
22 NET1 23 NET2	0 - 9	0 - 9	Local control NETWORK number for channels 1 and 2. If SET_NET1 or SET_NET2 =1, the network number is already set for the corresponding channel. In order to modify the already set network number, a COLD START of the AC 400 is necessary. All local control network numbers in the AC 400 must be unique. NET = 0 if port is not used.
25 SET_NET1 26 SET_NET2	0 - 1	0 - 1	If SET_NET1 or SET_NET2 =1, the network number has been set for the corresponding channel.
33 VALID1 42 VALID2	0 - 1	0 - 1	VALID1 = 1 when DSR1, CTS1, etc., are valid. VALID2 = 1 when DSR2, CTS2, etc., are valid.
34 DSR1 43 DSR2	0 - 1	0 - 1	Value read from signal Data Set Ready (CCITT V.24 107) for the corresponding channel.
35 CTS1 44 CTS2	0 - 1	0 - 1	Value read from signal Clear To Send (CCITT V.24 106) for the corresponding channel.
36 DCD1 45 DCD2	0 - 1	0 - 1	Value read from signal Data Carrier Detect (CCITT V.24 109) for the corresponding channel.
37 RI1 46 RI2	0 - 1	0 - 1	Value read from signal Ring Indicator (CCITT V.24125) for the corresponding channel.

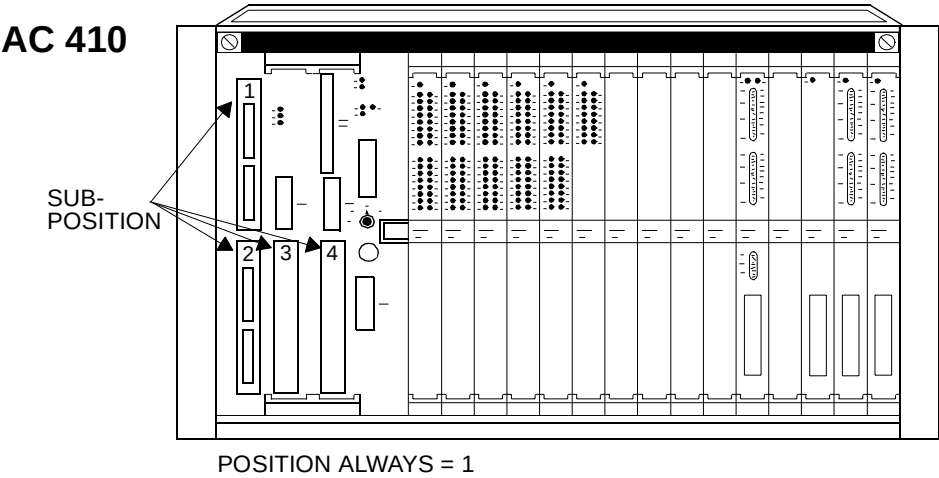


Figure 2-3. Position and Subposition for CI535 Submodule in Advant Controller 410

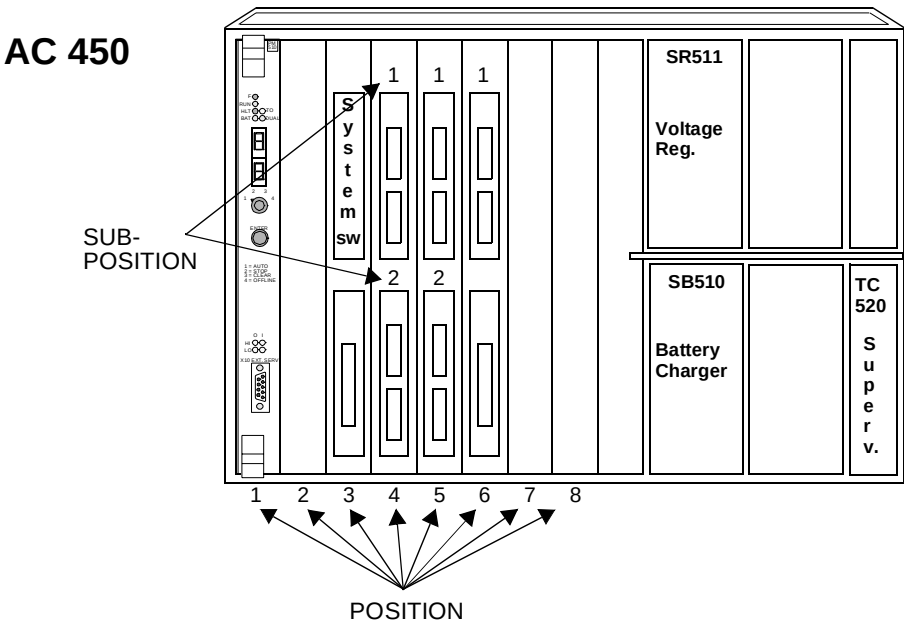


Figure 2-4. Position and Subposition for CI535 Submodule in Advant Controller 450

Cable for MVI Communication

The cable TK577 has a DE9 socket for connection to one port on the front of the CI535 submodule and a DE25 PIN for connection to the modem. The connection between the 9- and the 25-PIN connectors is shown in [Table 2-2](#). No wiring of the modem signals is made in the cable TK577. The cable TK595 has a DE9 socket for connection to one port of the CI535 submodule and a DE9 PIN for connection to the modem.

Table 2-2. Description of Connections for MVI Communication per CCITT V.24

Description	Signal	No. in 9-PIN Connector	No. in 25-PIN Connector	CCITT V.24	Direction
					IN ← — → OUT
Data Carrier Detect	DCD	1	8	109	←
Receive Data	RD	2	3	104	←
Transmit Data	TD	3	2	103	→
Data Terminal Ready	DTR	4	20	108/2	→
Signal Ground	SG	5	7	102	↔
Data Set Ready	DSR	6	6	107	←
Request To Send	RTS	7	4	105	→
Clear To Send	CTS	8	5	106	←
Ring Indicator	RI	9	22	125	←

If the port is set up for full duplex handling of modem signals, you can wire the cable TK57/TK595 as shown in [Figure 2-5](#). The setup of modem signals can also be strapped in the modems.

Description	Signal	No. in 9-PIN Connector at MVI-side	CCITT V.24
Request To Send	RTS	7 —	105
Clear To Send	CTS	8 ←	106
Data Set Ready	DSR	6 ←	107
Data Carrier Detect	DCD	1 ←	109
Data Terminal Ready	DTR	4 —	108/2

Figure 2-5. Wiring for Full Duplex Handling of Modem Signals.

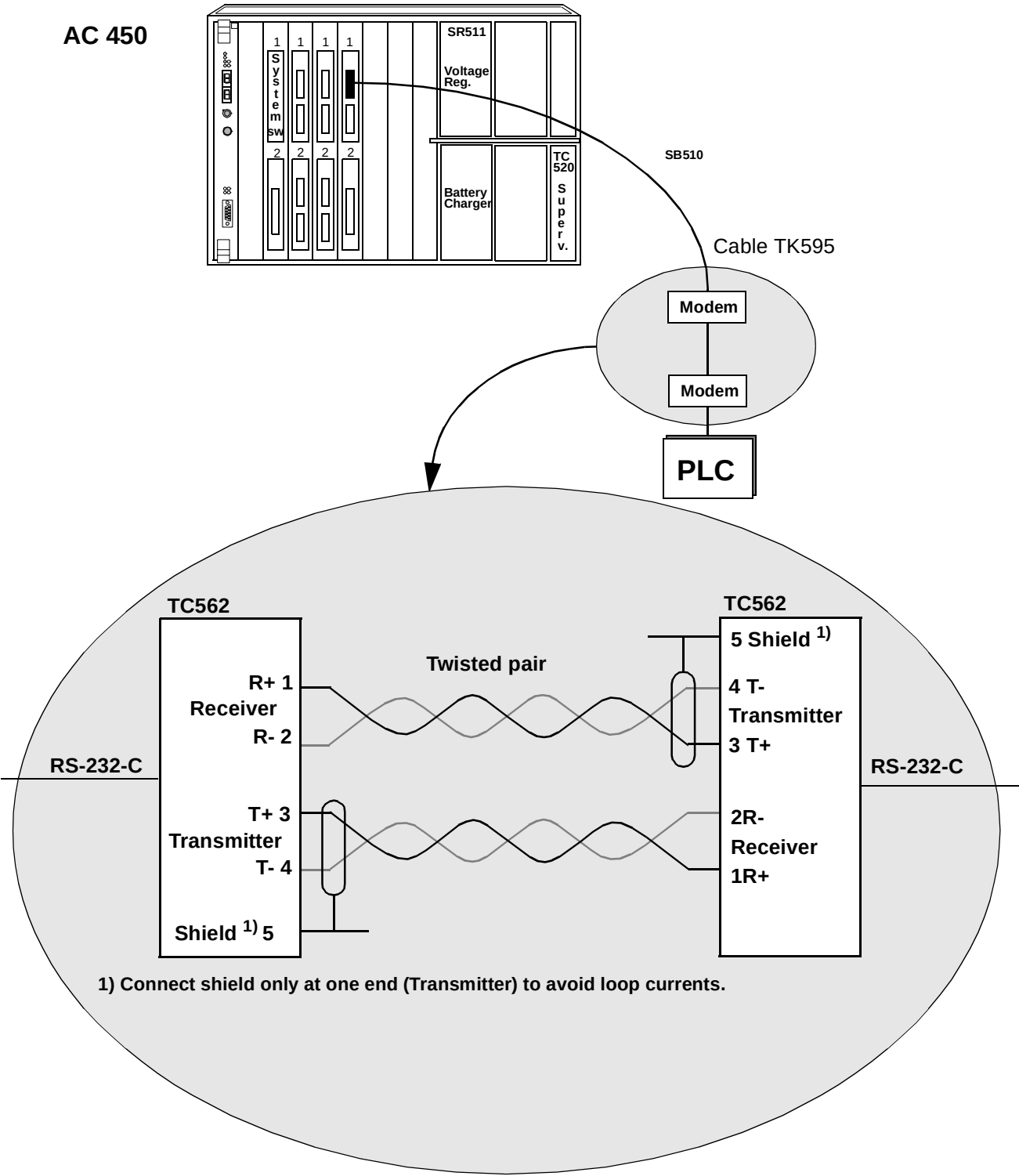


Figure 2-6. Cables and Modems in a Full Duplex Point-to-point Connection

Cable for SLIP Communication

The cable for SLIP communication has a 9-position D-Sub connector receptacle (female) for connection to one port on the HP9000/700 work station and a 9-position D-Sub connector receptacle (female) for connection to one port on the front of the CI535 submodule. The connection between the 9-PIN connectors is shown in [Table 2-3](#).

Table 2-3. Description of Connections for SLIP Communication per CCITT V.24

Description	Signal	No. in 9-PIN Connector 1	No. in 9-PIN Connector 2	CCITT V.24	Direction IN ← → OUT
Data Carrier Detect	DCD	1	1	109	←
Receive/Transmit Data ⁽¹⁾	RD/TD	2	3	104	← →
Transmit/Receive Data	TD/RD	3	2	103	→ ←
Data Terminal Ready	DTR	4	4	108/2	→
Signal Ground	SG	5	5	102	↔
Data Set Ready	DSR	6	6	107	←
Request To Send	RTS	7	7	105	→
Clear To Send	CTS	8	8	106	←
Ring Indicator	RI	9	9	125	←

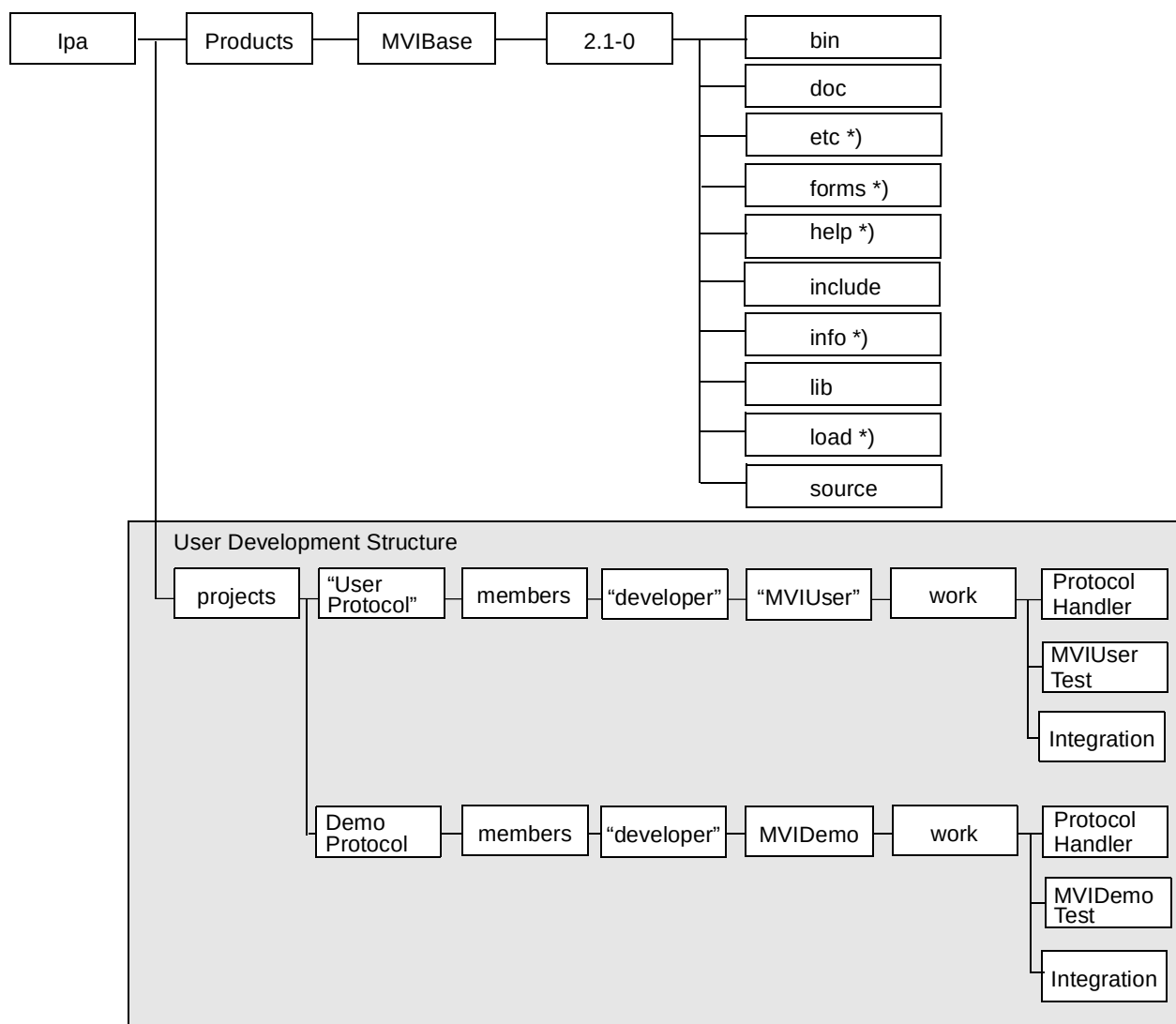
(1) **Note** that Receive Data and Transmit Data are crossed.

The cable should be wired for full duplex handling of modem signals, as shown in [Table 2-5](#).

2.2.2 Software

2.2.2.1 MVIBase and User Development Structure

MVIBase is installed in an SDE product structure and the User Development Structure is created using the SDE Project Manager. [Figure 2-7](#) shows the directory structure for the MVI Development Environment.



*) Not used in current version.

Figure 2-7. Directory Structure for the MVI Development Environment

The MVIBase product is delivered on a DAT tape in the update/fpkg format. For detailed information, see the manual *Creating Product Packages for HP-UX* and the manual page for **update**.

To install the MVIBase product, take the following steps:

1. Connect the DAT driver to the SCSI port and load the distribution tape.
2. Log in as root: **\$ su root**.
3. Run update: **\$ update**.

2.2.2.2 SDE

Installation of SDE consists of loading the software and setting up the configuration files. The installation procedures are described in the *SDE Installation Guide* found in the SDE product binder.

2.2.2.3 SLIP

Installation of PPL (SLIP interface) consists of loading the software. PPL software is supplied on the LAN/9000 installation tape. It is part of the SLIP-RUN file set. If you are loading the entire networking partition, this file set is loaded automatically. The installation procedures are described in *HP 9000 Networking, Installing and Administering LAN/9000 Software*.

2.3 Shut-down Procedures

Use normal logout procedure, that is, give the command *logout* at the command shell prompt, to shut down the communication between the development system and the target system.

Set the terminal SERVICE to 0 on the CI535 data base element to shut down the communication on the link. The communication is also stopped when the Advant Controller 400 is in configuration mode (mode P2, indicated on the front of the main processor module).

2.4 Start-up Procedures

Use normal login procedure, that is, give the command *rlogin <target>* at the HP-UX command shell prompt, to start up the communication between the development system and the target system.

To start up the communication link, you must set up the hardware and the data base element CI535, as described in [Section 2.2.1, Hardware](#).

The CI535 submodule always restarts if you restart the Advant Controller 400 in any way. The CI535 also restarts when the terminal SERVICE on the CI535 element is changed from 0 to 1.

Check all the system messages when you restart the CI535 submodule. All configuration errors are reported at start-up. Error messages with task name MVIxxx sent from the MVI Runtime Environment are translated in [Section 5.3, Error Messages](#).

NOTE

Some errors are only reported once.

2.5 Product Verification

When the CI535 submodule is correctly installed and the corresponding CI535 data base element is filled in, the following is indicated:

- The green LED indicator on the front of the CI535 submodule is lit.
- Data base element CI535 Terminals ERR and ERRTYPE = 0.

The CI535 element shows the following when the link is connected and the GVD (Get Values Dynamically) command is given:

- DSR = 1
- CTS = 1 (full duplex) or toggling between 1 and 0 (half duplex)
- DCD= 1 (full duplex) or toggling between 1 and 0 (half duplex).

Chapter 3 Configuration/Application Building

3.1 Design Consideration

Before starting with the design, your first consideration is to investigate if the protocol has a well-defined Data Link layer. If it does, then consider whether or not the Protocol Handler should be split into two tasks:

- Network and/or Function Handler
- Data Link Handler.

If, on the other hand, no distinct Data Link layer exists in the protocol, include the Protocol Handler function in one task.

If both slave and master modes of execution are going to be implemented (for master/slave protocol types), these two modes should be handled by two separate functions, sharing a number of common functions.

The Host Interface is implemented to isolate the Host System (process station) from the Protocol Handler. The MVI Protocol Handler Interface is used for communication between the Protocol Handler and the Host Interface. The MVI Protocol Handler Interface is described in [Section 3.6.5, MVI Protocol Handler Interface](#).

If the Host communication is based on MVI Data Set (MS), use the Standard Stub as Host Interface which is included in the MVI Environment. We strongly recommend that you use the Standard Stub to secure future compatibility. If, on the other hand, you need to use some other kind of Host Interface, a Host Interface which maps between the Host System and the MVI Protocol Handler Interface must be implemented.

A start-up function is implemented to create devices used by the MVI Runtime Environment, the Protocol Handler and the Host Interface. The start-up function also spawns the Host Interface task, which in turn spawns the Protocol Handler task.

A Protocol Handler is normally divided into a number of different parts:

- Initialization
 - Initialize protocol variables and data structures
- Setup
 - Handle setup information from host system and initialize devices such as the UART
- Runtime master
 - Main loop for master
- Runtime slave
 - Main loop for slave
- Error handling
 - Handling of various error conditions

- Termination
 - Release all allocated resources.

There are two alternate designs possible for the Protocol Handler design:

- State machine, where each state handles one of all possible events.
- Conventional multiple handling switch statement, handling all possible events.

The convention for task names on the CI535 communication submodule, which must be followed, is described in [Figure 3-1](#).

```
Task Name = tMvixxxnn
xxx = Hdl for the Protocol Handler or Network Handler
xxx = DLL for the Protocol Handler Data Link Handler

xxx = Stb for the Host Interface
xxx = Mux for the Protocol Multiplexor
xxx = Sup for the Startup function

nn = 01 for Task Instance One
nn = 02 for Task Instance Two
```

Figure 3-1. MVI Task Name Convention

The reason for using this convention is that the task name is part of the system messages which are sent from the CI535 submodule (using mviSysmess) to the Advant Controller 400 Series process station. They translate to the system message format used in the process station. This translation is described in [Section 5.3, Error Messages](#).

3.2 Capacity & Performance

Capacity and performance depends on implemented protocol. A general rough estimation is that you can use 70 percent of the communication speed for messages. With a message header and trailer overhead of 20 percent, the data transfer rate, in characters/second, would be:
 $(0.7/1.2) * (\text{speed in bps}) / (\# \text{ of bits per char})$.

3.3 MVI Development Environment Setup

For you to operate the MVI Development Environment, you must modify or create a number of setup parameters and files, as described below.

3.3.1 SDE

If you use SDE, the project must be set up in SDE. That is, the project must be created - a system and possibly one, or several, subsystems must be created. For details, please refer to the *SDE Product Binder*.

3.3.2 MVIBase

VxWorks development tools must be configured by definition of certain environment variables, that is, **MANPATH**, **VX_CPU_FAMILY** and **GCC_EXEC_PREFIX**. Please refer to the *GNU Toolkit User's Guide*. The path to the VxWorks development tools must be added to the **PATH** environment variable. Definition of these variables is preferably done in *.kshrc* (korn shell start-up file) in the developer's login directory. See [Figure 3-2](#).

```
# Environment script for Korn Shell.
# This script is executed for each new Korn Shell.
:
:
# Set up the path to development tools
PATH=$PATH:/ipa/products/MVIBase/2.1-0/bin
export PATH
# Set up path to manual pages
MANPATH=$MANPATH:/ipa/products/MVIBase/2.1-0/doc
export MANPATH
#Define variables for GNU tools
VX_CPU_FAMILY=68k
export VX_CPU_FAMILY
GCC_EXEC_PREFIX=/ipa/products/MVIBase/2.1-0/bin
export GCC_EXEC_PREFIX
VX_HOSTTYPE=hp9700
export VX_HOSTTYPE
```

Figure 3-2. A Sample *.kshrc* File

3.3.3 Host/Target Internet Addresses

The Internet addresses used for host and target must be legal addresses, that is, addresses owned by the corporation or organization that uses them. Using an illegal Internet address may cause the owner of the address to lose data packages.

See your company's policy regarding the use of Internet addresses. Your work station system administrator should be able to help you.

3.3.3.1 Configuring the Host System

Before the point-to-point SLIP network can be used, the network software on the host node must be correctly configured. There are three main tasks in configuring the host network software:

- Initializing the host network software.
- Establishing the target system name and network address on the host.
- Giving the target system appropriate access permissions on the host.

Initializing the Host Network Software

The host system (HP9000/700 work station) automatically initialize the network subsystem and activate network processes. This typically includes configuring the network interface and starting various network daemons. In addition to this, the SLIP interface have to be configured and initialized. This is described in [Section 3.3.4, SLIP Interface](#).

Establishing the Target System Name and Address: /etc/hosts

The host system maintains a data base of the names and network addresses of the systems accessible from the local system. This data base is kept in the ASCII file /etc/hosts which contains a line for each system. Each line consists of an Internet address and the name(s) of the system at that address. This file must have entries for your host system and the target system.

For example, suppose your host system is called slip_host and has been assigned Internet address 138.221.124.9, and your target system is called slip_target and has been assigned Internet address 138.221.124.6. The file /etc/hosts should then contain the following lines:

```
138.221.124.9slip_host
138.221.124.6slip_target
```

NOTE

If your host system is running the Network Information Service (NIS) and/or Domain Name Services, then the “hosts data base” is maintained by the Network Information Service facilities or the Domain Name Services facilities that are beyond the scope of this document. Consult your system administrator if you are running the Network Information Service and/or Domain Name Services.

Giving the Target System Access to the Host: .rhosts and /etc/hosts.equiv

The host system restricts network access via remote login, remote command execution and remote file access. This is done with the .rhosts file in the users home directory, and more globally with the /etc/hosts.equiv file. The .rhosts file contains a list of systems that have access to your login. Thus, in our example above, to allow the target system to log in with your user name and access files with your permissions, you would create a .rhosts file in your home directory, containing the line:

```
slip_target
```

The file /etc/hosts.equiv provides a less selective mechanism. System listed in this file are allowed login access to any user defined in the local system (except the super-user root). Thus, adding the target system name to /etc/hosts.equiv will allow the target system to log in with any user name on the local system.

When inetd (Internet super-server, which invokes Internet server processes as needed) accepts a connection from a remote system, it checks the address of the remote host requesting the service against the list of remote hosts to be allowed or denied access to the specific service. The file inetd.sec allows the system administrator to control which hosts (or networks in general) are allowed to use the system remotely. This file constitutes an extra layer of security in addition to the normal checks done by the services. It precedes the security of the servers; that is, a server is not started by the Internet daemon unless the host requesting the service is a valid host according to inetd.sec.

If file /usr/adm/inetd.sec, does not exist, security is limited to that implemented by the servers. Changes to inetd.sec apply to any subsequent connections.

The lines in the file contain a service name, permission field, and the Internet addresses or official names of the hosts and networks allowed to use that service in the local host.

The fields in each line are as follows:

<service name> <allow|deny> <host or net addresses, host or net names>

where:

- Service name is the name (not alias) of a valid service in file /etc/services. The service name for RPC-based services (NFS) is the name (not alias) of a valid service in file /etc/rpc. A service name in /etc/rpc corresponds to a unique RPC program number.
- Allow|deny determines whether the list of remote hosts in the next field is allowed or denied access to the specified service. Multiple allow|deny lines for each service is unsupported. If there are multiple allow|deny lines for a particular service, all but the last line are ignored
- Addresses and names are separated by white space. Any mix of addresses and names is allowed. To continue a line, terminate it with \.

Thus, in our example above, to allow the target system to access the host system using the services shell (remote shell), login (remote login) and rexd (remote procedure calls) you would add the following lines to the file /usr/adm/inetd.sec:

```
shell  allowxxx.xxx.xxxslip_target
login  allowxxx.xxx.xxxslip_target
rex    allowxxx.xxx.xxxslip_target
```

NOTE

These services might already be defined for different remote nodes or subnets. In that case, add the name of the target system (slip_target) to the lines.

3.3.3.2 Configuring the Target System

The host system and target system Internet addresses must also be configured in the target system. This is done by editing the file inet.c, prior to generating a VxBase system. The file inet.c is described in [Section 3.3.8, Preparation for System Building](#).

3.3.4 SLIP Interface

Configuration of PPL consists of setting up necessary support files. These procedures require super-user privileges.

- The serial port on the HP 9000/700 work station, used for SLIP, requires a character device file entry with major number 1 (one) and minor number set up for direct connect on the serial port of your choice. If this file does not exist with proper values for major and minor numbers, you must create one. An example of how to create a device file using **mknod** follows:
\$ mknod /dev/tty00 c 1 0x204004¹
- Ensure that you have a /dev/ni device file. The PPL program uses this device to pass packets between the serial line and IP layer. If it is not present, create the device file using the following command:
\$ mknod /dev/ni c 56 0

1. The minor number value depends on a number of details concerning your HP9000/700 work station, such as port address type. Consult your HP9000/700 system administrator for details.

PPL must be configured and initialized if you wish to use the SLIP protocol, the Internet address for the target system and the serial port available on the HP 9000/700 work station. PPL configuration involves creation of three files:

- /usr/lib/ppl/ppl.users
- /usr/lib/ppl/ppl.ipool
- /usr/lib/ppl/ppl.remotes.

For direct connection, which is the only configuration used in the MVI Development Environment, ppl.remotes is the only file required and is the only file described here. The ppl.remotes file is a simple ASCII file that the super-user owns (creates). A template exists for the ppl.remotes file in the **/usr/lib/ppl** directory.

The ppl.remotes file is the central data base used by the PPL program. The file contains an entry for each remote host to which PPL can establish a connection. Each entry consists of a multi-line form. Depending on the type of connection you are permitting, that is, direct, dial-in, dial-out, or both dial-in and dial-out, you enter different data. However, the format of the form is the same for all types, see [Figure 3-3](#).

```
1) # your comments here
2) slip_target      # remote hostname or Internet address
3) slip_host       # local hostname or Internet address
4) 255.255.255.0   # Internet mask
5) SLIP            # protocol SLIP, ASLIPC, ASLIPS, PPP
6) DIRECT          # type DIRECT, DIALIN, DIALOUT, DIALIN & DIALOUT
7)                # UUCP system name
8) NONE           # line parity EVEN, ODD, NONE
9) 19200           # line speed
10) /dev/tty00     # serial line
11)               # phone number
12) NO             # modem control available YES, NO
13)               # log in info
14)               # command name
```

Figure 3-3. ppl.remotes Form Format

The following is a description of each data field in the ppl.remotes form:

1. The first line is a comment line, describing the entry. If necessary, use additional lines for comments. Begin each line with a "#".
2. This field identifies the remote host. It may specify a host name or alias, provided the name or alias can be resolved to an Internet address (by /etc/hosts, the Yellow Pages or domain name server). It may also be the Internet address in dot notation. This field is required.

3. This field identifies the local host. It may specify a host name or alias, provided the name or alias can be resolved to an Internet address (by /etc/hosts, the Yellow Pages or domain name server). It may also be the Internet address in dot notation. This field is required.
4. This field specifies a mask which may be used to set up an IP subnet. If left blank, the mask is derived from the local Internet address.
5. This field specifies the protocol to be used for connection, i.e., Serial Line Internet Protocol (SLIP), Abbreviated Serial Line Internet Protocol Client (ASLIPC), Abbreviated Serial Line Internet Protocol Server (ASLIPS) or Point-to-Point Protocol (PPP). Of these protocols, only SLIP is used by the MVI Development Environment. This field is required.
6. This field specifies the connection type. Only direct connection is used by the MVI Development Environment. This field is required.
7. This field specifies the UUCP system name. It is not used by the MVI Development Environment.
8. This field specifies line parity and is only used for dial-up connections. It is not used by the MVI Development Environment.
9. This field specifies the line speed (baud rate) to be used over the connection. The supported baud rate is limited by the system hardware. This field is required.
10. This field specifies the device file to be used for the connection. This field is required.
11. Phone number for the remote host. This field is not used by the MVI Development Environment.
12. This field controls the use of the modem signals during and after the connect phase of PPL. This field is required and must be set to NO.
13. This field supplies the dialogue needed to log in to the host. This field is not used by the MVI Development Environment.
14. This field supplies the dialogue needed to invoke SLIP on the connected host. This field is not used by the MVI Development Environment.

The HP 9000 system administrator may set up a direct PPL connection, which can be initialized at boot time or any time thereafter. To set up a direct connection at boot time, the super-user inserts a line in the /etc/netlinkrc script like the following:

ppl -o slip_target

where slip_target is the name of the target. As mentioned above, the command can be issued at any time from the command shell:

\$ ppl -o slip_target.

For more information about PPL, see *HP 9000 Networking, Using Serial Line IP Protocols*.

NOTE

The host system and target system should be connected via the serial port which you have chosen for your HP 9000/700 work station and channel two on the CI536 submodule (see [Appendix D, Hardware Module](#)). A **getty**¹ must not be running on the serial port chosen for the HP 9000/700 work station.

1. A process invoked by init to set terminal type, line mode speed and line discipline. It is the second process in the series (init, getty, login, shell) that ultimately connects a user with the HP-UX system. For details, consult your HP 9000/700 system administrator.

3.3.5 Test Directory

Create a test directory in the user Development Environment, `/ipa/projects/"UserProtocol"/members/"developer"/"MVIUser"/work/MVIUserTest`, from which the modules are downloaded during the development phase. This directory should contain links to the object modules created when compiling the Protocol Handler in the developer's SDE work structure:

- **\$ ln -s /ipa/projects/"UserProtocol"/members/"developer"/"MVIUser"/work/ProtocolHandler/"protocol module".do "protocol module".do**

NOTE

In the path above, "developer" refers to the login id of the person that develops the communication protocol. See also [Figure 2-7](#).

3.3.6 Download Script

For convenience, create a download script in the test directory, which is executed in the target to download the modules under development. See [Figure 3-4](#).

```
ld < "protocol module1".do
ld < "protocol module2".do
:
ld < "protocol moduleN".do
```

Figure 3-4. A Sample Download Script

This is an ordinary text file which is executed from the target shell command line. Before executing the script, you must set the user and working directory where the script can be found. This is done with the commands:

```
-> iam "developer"
-> cd "slip_host:/ipa/projects/MVIUser/members/"developer"/"UserProtocol"/work/
MVIUserTest"
-> < downloadScript
```

3.3.7 Debugger Initialization

For convenience, create a start-up file, `.vxgdbinit`, in the test directory, which is executed by VxGDB during start-up. In this file, the developer can define paths to the source code to be debugged, initial breakpoints, the target to which to connect, etc. See [Figure 3-5](#).

```
target vxworks slip_target
dir /ipa/projects/"UserProtocol"/members/"developer"/"MVIUser"/work/ProtocolHandler
dir /ipa/products/MVIBase/1.0-0/include
break protocolHandlerMain
run protocolHandlerMain 1
```

Figure 3-5. A Sample `.vxgdbinit` File

In order to work, VxGDB must find the symbol table for the load module, resident in Flash PROM, in the target. The way to ensure this is to define a softlink named `vxWorks.st` in the developer's login directory, pointing to the load module generated in `/ipa/projects/"UserProtocol"/members/"developer"/work/Integration`. The softlink is created with the following command:

```
$ cd
$ ln -s /ipa/projects/
"UserProtocol"/members/"developer"/work/Integration/
vwci535do.st vxWorks.st
```

3.3.8 Preparation for System Building

In the Integration subsystem directory (`/ipa/projects/"UserProtocol"/members/"developer"/"MVIUser"/work/Integration`), make a copy of the Makefile for generating a PROMed system (`makesystem.mk`), `initAppCI535.c`, `inet.c`, `dbskonfi.c`, `dbskonfi_inc.h` and the PROMed system generation scripts (`mkProdSystem` and `mkTestSystem`) using the commands:

```
$ cp /ipa/products/MVIBase/2.1-0/source/MakeSystem.mk ./Makefile
$ cp /ipa/products/MVIBase/2.1-0/source/initAppCI535.c .
$ cp /ipa/products/MVIBase/2.1-0/source/inet.c .
$ cp /ipa/products/MVIBase/2.1-0/source/dbskonfi.c .
$ cp /ipa/products/MVIBase/2.1-0/source/dbskonfi_inc.h .
$ cp /ipa/products/MVIBase/2.1-0/source/mkProdSystem .
$ cp /ipa/products/MVIBase/2.1-0/source/mkTestSystem .
```

NOTE

The dot at the end of the above commands denotes the current directory.

The **Makefile** is used together with the system generation scripts (`mkProdSystem` and `mkTestSystem`) to build a PROMable system.

The **initAppCI535.c** module initializes the Host System Interface and starts the MVI start-up task. See [Figure 3-6](#).

```
extern void mviStartupMain(void);
STATUS initApp(void)
{
#ifdef INCLUDE_TRANSCIEIVER
    transceiverInit(60, 120, 160);
    tranChanDrv();
#endif /* INCLUDE_TRANSCIEIVER */
    vxsmInitPointers();
    taskSpawn("tmviSup", 50, 0, 2000,
              mviStartupMain, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    return (OK);
}
```

Figure 3-6. Init Application Function

You may need to change the `transceiverInit()` call to change the number of transceiver buffers in the buffer pool or the size of the transceiver buffers. See [Section 3.6.8.4, TransceiverInit](#).

The **inet.c** module defines two strings, `inetHostAddr` and `inetTargetAddr`, which must be edited to hold the Internet addresses that is to be used for Host System and Target System. For information on how to select these addresses, see [Section 3.3.3, Host/Target Internet Addresses](#).

```
char* inetHostAddr    = "138.221.124.9";
char* inetTargetAddr  = "138.221.124.6";
```

Figure 3-7. Internet Addresses Configuration Function

The **dbskonfi.c** and **dbskonfi_inc.h** modules initializes part of the Host System Interface. You should not need to change these modules.

The **mkTestSystem** module is a script which is used for building a Development System. This kind of system includes support for networking, SLIP and remote debugging. It uses the second RS-232-C communication interface (channel two) for SLIP communication. This means that only one channel is available for MVI communication. You should not need to change this module.

The **mkProdSystem** module is a script which is used for building a Production System. This kind of system excludes networking, SLIP and remote debugging, but includes the protocol developed. This script refers to the module that implements the MVI Demo Protocol Handler. This reference has to be changed to refer to the Protocol Developed.

```
make -f Makefile \
LIB_EXTRA="\
/ipa/products/MVIBase/2.1-0/lib/mviChkSumLib.o \
/ipa/products/MVIBase/2.1-0/lib/mviDLLib.o \
/ipa/products/MVIBase/2.1-0/lib/mviProtInt.o \
/ipa/products/MVIBase/2.1-0/lib/mviStartup.o \
/ipa/products/MVIBase/2.1-0/lib/mviFBStubCommon.o \
/ipa/products/MVIBase/2.1-0/lib/mviFBStubMux.o \
/ipa/products/MVIBase/2.1-0/lib/ac110Stub.o \
/ipa/products/MVIBase/2.1-0/lib/ac110DevSpecSoft.o \
/ipa/products/MVIBase/2.1-0/lib/mviAC400StandardStub.o \
/ipa/products/MVIBase/2.1-0/lib/mviAC400StandardMain.o \
/ipa/products/MVIBase/2.1-0/lib/mviDemo.o \
/ipa/products/MVIBase/2.1-0/lib/rcomEvMsgTransf.o " vwc15350.st.hex
```

Figure 3-8. System Generation Script (mkProdSystem)

3.4 Tutorial

This section contains a tutorial on how to implement PLC/RTU protocols in the MVI Development Environment. The code for this tutorial, found in [Appendix A, Code Example](#), is not complete. The complete code is found in the Demo Protocol Handler included in the MVI Development Environment. For a detailed description of SDE and SDE related commands, refer to the *SDE Product Binder*.

NOTE

This tutorial assumes that MVIBase is installed according to [Section 2.2.2.1, MVIBase and User Development Structure](#) and set up according to [Section 3.3.2, MVIBase](#).

Create Environment

1. Start the SDE Project Manager using the **projm** command in a terminal window, and start the SDE System Manager using the **system** command in a terminal window.
2. Create the DemoProtocol project using the SDE Project Manager.
3. Create the MVIDemo system using the SDE System Manager and add the system to the DemoProtocol project using the SDE Project Manager.
4. Create the Protocol Handler and Integration subsystems using the SDE System Manager.
5. Add the MVIDemo system to the DemoProtocol project using the SDE Project Manager.
6. Use “SDE” command from “Add” menu, start “Add Project Session ...” and add your project session. The first time you select the project, invoke “Create Project Directories ...” from selected menu. This means that the work directories for system and subsystems are created automatically, the first time **project set** is done.

NOTE

If there are several members, start “Add New Member” dialog box from Admin-menu and define all members of the project.

7. On configuration level (/ipa/projects/DemoProtocol/members/"developer"/MVIDemo/work/), make a copy of the makefiles import.mk, export.mk and internals.mk using the commands:

```
$ cp /ipa/products/MVIBase/2.1-0/source/imports.mkf .  
$ cp /ipa/products/MVIBase/2.1-0/source/exports.mkf .  
$ cp /ipa/products/MVIBase/2.1-0/source/internals.mkf .
```
8. In the Integration subsystem directory (/ipa/projects/DemoProtocol/members/"developer"/MVIDemo/work/Integration), make a copy of the PROMed system Makefile (makesystem.mk), initAppCI535.c, inet.c, dbsconfi.c, dbsconfi_inc.h and the PROMed system generation scripts (mkProdSystem and mkTestSystem) using the commands:

```
$ cd ../Integration  
$ cp /ipa/products/MVIBase/2.1-0/source/MakeSystem.mk  
./Makefile  
$ cp /ipa/products/MVIBase/2.1-0/source/initAppCI535.c .  
$ cp /ipa/products/MVIBase/2.1-0/source/inet.c .  
$ cp /ipa/products/MVIBase/2.1-0/source/dbsconfi.c .  
$ cp /ipa/products/MVIBase/2.1-0/source/dbsconfi_inc.h .  
$ cp /ipa/products/MVIBase/2.1-0/source/mkProdSystem .  
$ cp /ipa/products/MVIBase/2.1-0/source/mkTestSystem .
```
9. Check that the system generation scripts are executable. If they are not, change the access rights using the command:

```
$ chmod ug+x mkTestSystem mkProdSystem
```

NOTE

The dot at the end of the above commands denotes the current directory.

Generate a PROMed Test System

10. Edit the file inet.c. Host and Target Internet addresses for the point-to-point SLIP network must be specified in this file. How to select Internet addresses is described in [Section 3.3.3, Host/Target Internet Addresses](#).
11. Generate a PROMed test system using the command:

```
$ mkTestSystem
```

This command generates a MVI system load image VxWorks.st (link to vwci535do.st) without the MVIDemo Protocol Handler, which will be dynamically downloaded. It also generates a Motorola S-record file (vwci535do.st.hex) of the load image, which is used by the Flash PROM programmer.
12. Download the S-record file to the Flash PROM programmer and make the PROMs.
13. Install the four Flash PROMs on the CI536 submodule, placed as shown in [Figure D-2](#).

Build Object Modules

14. Copy the sample source code and Makefile for the MVIDemo Protocol Handler to the ProtocolHandler subsystem using the commands:

```
$ cd ../ProtocolHandler
$ cp /ipa/products/MVIBase/2.1-0/source/mviDemo.c .
$ cp /ipa/products/MVIBase/2.1-0/source/mviDemo.h .
$ cp /ipa/products/MVIBase/2.1-0/source/Makefile .
```

These commands copies the modules mviDemo.c, mviDemo.h and Makefile to the working directory.

NOTE

The dot at the end of the above commands denotes the current directory.

15. Enter the module mviDemo.c into the editor and go to the function mviDemoMain. Go through the annotated code example in [Appendix A, Code Example](#) together with the source code for the mviDemo Protocol Handler entered in the editor.
16. Build the object module from the source code using the SDE/SoftBench Development Manager and Program Builder. Start the Program Builder, fill in "all.d" in the Build Options field and press the Build Push Button. This will give the module mviDemo.do.

Build Application in the Advant Controller 400 Process Station

17. Fill in the values in the CI535 data base element as described in [Section 2.2.1, Hardware](#) and [Appendix B, MVI Demo Protocol Configuration](#).
18. Build Line Characteristics MS, Network Configuration MS, RTU Status MS and Register Address MS as described in [Section 3.6.7.2, Command MS used in Master Mode](#). and [Appendix B, MVI Demo Protocol Configuration](#).
19. Build Data MS for data transfer as described in [Section 3.6.7.3, Data MS for Data Transfer](#). and [Appendix B, MVI Demo Protocol Configuration](#).
20. Build the PC program for data flow control using the type circuits described in [Section 3.6.7.4, Type Circuits for MVI Protocol Data Flow Control](#). and [Appendix B, MVI Demo Protocol Configuration](#).
21. Set the Advant Controller 400 process station in Operation mode using the command **diconfig**.

Download to Target and Start MVI Runtime Environment

22. Change directory to the test directory using the command:

```
$ cd /ipa/projects/DemoProtocol/members/"developer"/MVIDemo/
work/ MVIDemoTest
```

23. Create a link to the object module created in the development directory using the command:

```
$ ln -s /ipa/projects/DemoProtocol/members/"developer"/
MVIDemo/work/ProtocolHandler/mviDemo.do mviDemo.do
```

24. Initiate the slip connection between the host and target using the command:

```
$ ppl -o slip_target.
```
25. Log in to the target module using the command:

```
rlogin slip_target.
```
26. From the target shell, define user by the command:

```
-> iam "<developer>"
```


and change directory using the command:

```
-> cd "slip_host:/ipa/projects/DemoProtocol/members/  
"developer"/MVIDemo/ work/MVIDemoTest"
```
27. Download the Protocol Handler object module using the command:

```
-> ld < mviDemo.do
```

Debug Code

28. From another window with the MVIUserTest directory as current directory, invoke the debugger using the command:

```
$ vxgdb &
```
29. Now you must initiate the debugger. Do this from the VxGDB command line.
 - a. Connect to the target to be debugged using the command:

```
(vxgdb &) target vxworks slip_target
```
 - b. Define the directory where the source code to be debugged is found using the command:

```
(vxgdb &) dir /ipa/projects/DemoProtocol/members/  
"developer"/MVIDemo/work/ProtocolHandler
```
 - c. Define the initial breakpoint from which to start debugging, using the command:

```
(vxgdb &) break protHndlMain
```
 - d. Start up the debugger using the command:

```
(vxgdb &) run protHndlMain 1
```
30. The debugger is now initiated and ready to run the mviDemo Protocol Handler from the initial breakpoint.

3.5 Application Procedures

3.5.1 Protocol Handler Development

This is a summary of the steps to take to implement a communication protocol using MVI. See [Section 3.6, Configuration/Application Building Menus](#) for a detailed description of the functions comprising the MVI Runtime Environment.

If you are using SDE, install it and set it up. See [Section 2.2.2.2, SDE](#) for installation instructions and [Section 3.3.1, SDE](#) for information about initialization of SDE. This is essential for steps 1 and 2 below.

Install and set up MVIBase. See [Section 2.2.2.1, MVIBase and User Development Structure](#) for installation instructions and [Section 3.3.2, MVIBase](#) for information about initialization of MVIBase. This is essential for steps 2, 3, 4 and 5 below.

Install and initialize PPL (SLIP). See [Section 2.2.2.3, SLIP](#) for installation instructions and [Section 3.3.4, SLIP Interface](#) for information on initialization of PPL. This is essential for steps 3 and 4 below.

Generate a PROMed development system. PROM the system generated and install the PROMs on the CI536 module. You need a PROM programmer with support for 28F020 type Flash PROMs.

The steps below correspond to the numbers in [Figure 3-9](#).

1. Create source code
 - Select a proper editor. The SoftBench editor (softedit) is first choice, but other editors such as emacs or vi will do.
2. Compile the source code
 - You must generate a makefile, or use the makefiles provided for the MVIDemo Protocol Handler.
3. Download to target
 - Select Internet addresses for host and target machines, see [Section 3.3.4, SLIP Interface](#).
 - You can use a download script, see [Section 3.3.6, Download Script](#).
4. Debug in target
 - Initialize the remote debugger. You can use a startup script, see [Section 3.3.7, Debugger Initialization](#).
5. Generate a PROMed production system.
6. PROM the system generated and install the PROMs on the CI535 module
 - You need a PROM programmer with support for 28F020 type Flash PROMs.

NOTE

This manual does not describe in detail how to use SDE tools. For a detailed description of SDE, refer to the *SDE Product binder*.

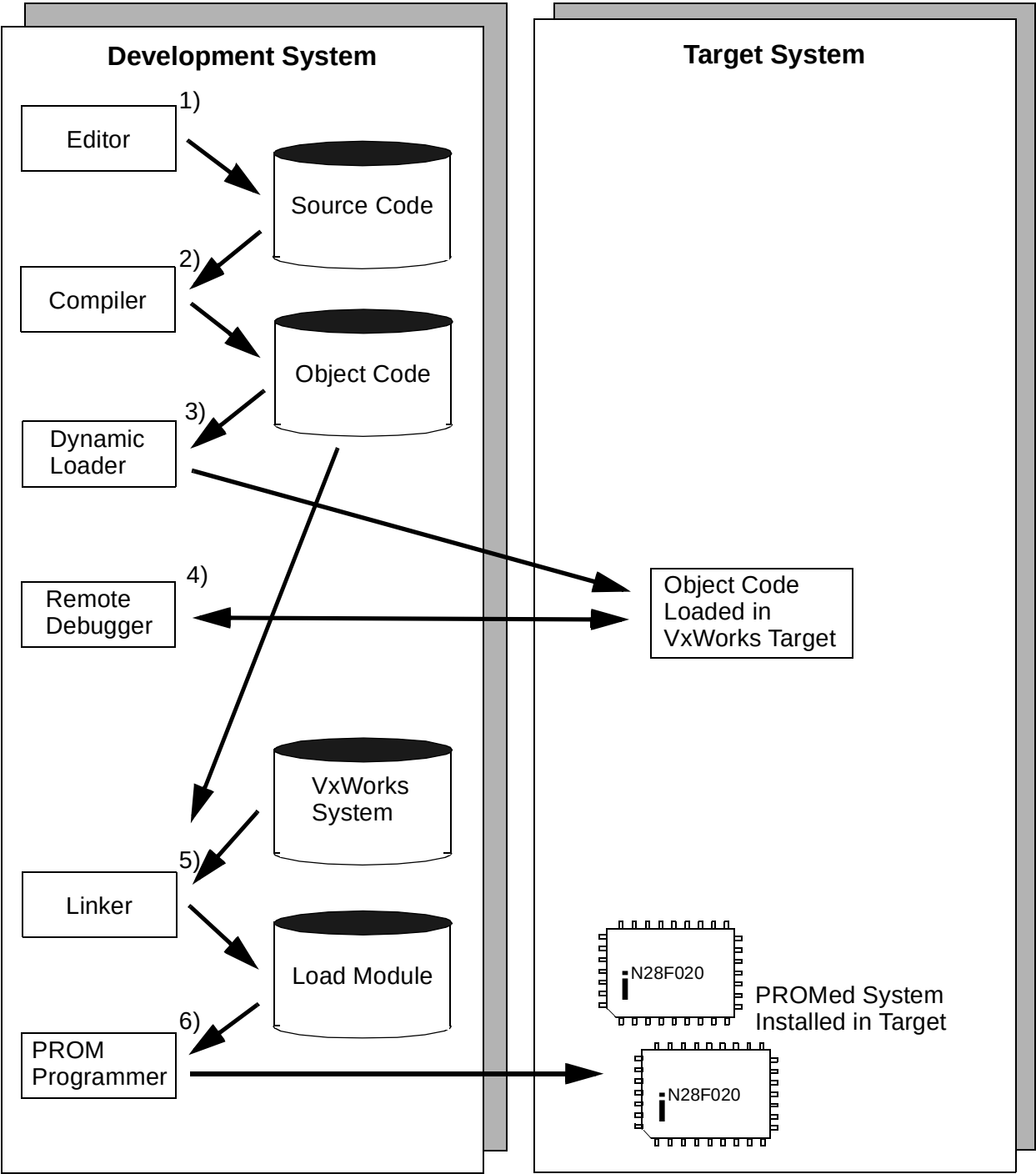


Figure 3-9. Protocol Handler Implementation Steps

3.5.1.1 Create Source Code

Consider the items mentioned in [Section 3.1, Design Consideration](#) before you start to implement the protocol. That is:

- Consider the Host communication, the Standard Stub and the MVI Protocol Handler Interface when deciding if and how to implement the Host Interface. The MVI Protocol Handler Interface is described in [Section 3.6.5, MVI Protocol Handler Interface](#). In most cases, you can use the Standard Stub as Host Interface, provided in MVI. If it is not possible to use the Standard Stub, then implement your own.
- Should the Protocol Handler be divided into two tasks, Network/Function Handler and Data Link Handler?
- Should both master and slave mode be implemented (for master/slave type protocols)?
- Should the Protocol Handler be implemented as a state machine or with a conventional multiple handling switch statement? The MVI Demo Protocol is an example of a Protocol Handler implemented using the state machine model.

If you are using SDE, take the following steps:

1. Start the SDE Project Manager using the **projm** command in a terminal window and start the SDE System Manager using the **sysm** command in a terminal window.
2. Create the project UserProtocol (or any other name of your choice) using the SDE Project Manager.
3. Create the system MVIUser (or any other name of your choice) using the SDE System Manager, and add the system to the project created using the SDE Project Manager.
4. Create the ProtocolHandler and Integration subsystems using the SDE System Manager.
5. Use the **project** command under the **set** menu in the SDE Project Manager to set the project. The first time you set the project, you should press the "Create work directories for all subsystems" Radio Button. This means that the work directories for system and subsystems are created automatically, the first time **project set** is done.
6. In the Protocol Handler subsystem directory, create the source code using the editor of your choice.

If you are not using SDE, you are on your own as far as project handling and source code handling are concerned. Take the following steps:

1. Create your working directory structure.
2. Create the source code using the editor of your choice.

3.5.1.2 Compile the Source Code

If you are using SDE, take the following steps.

1. On configuration level (/ipa/projects/"UserProtocol"/members/"developer"/"MVIUser"/work/), make copy of the makefiles import.mk, export.mk and internals.mk using the commands:

```
$ cp /ipa/products/MVIBase2.1-0/source/import.mk .  
$ cp /ipa/products/MVIBase/2.1-0/source/export.mk .  
$ cp /ipa/products/MVIBase/2.1-0/source/internals.mk .
```

The files export.mk and internals.mk refers to MVIDemo files and must be adopted for the protocol developed.

2. Copy the Makefile for the MVIDemo Protocol Handler to the ProtocolHandler subsystem using the commands:

```
$ cd ../ProtocolHandler
$ cp /ipa/products/MVIBase/2.1-0/source/Makefile .
```

This file refers to MVIDemo files and must be adopted for the protocol developed.

3. Use the SDE/SoftBench program Builder to build the object module(s). Start the Program Builder, fill in "all.d" in the Build Options field and press the Build Push Button. This will give the module "protocol module".do.
4. Correct any remaining errors using the editor.
5. Repeat steps 2 and 3 until the object module(s) are correctly built.

If you are not using SDE, take the following steps.

1. Generate a makefile. Use the following command to do this, with the current directory set to the directory containing the source code:

```
$ mkmf
```

Note that the *mkmf* program scans all files in the current directory, so the directory should contain only the source code and include files that implement the protocol. For an example of a makefile, see [Appendix C, A Makefile Example](#).

2. Build the object module(s) using the following command:

```
$ make
```

3. Correct any remaining errors using the editor.
4. Repeat steps 2 and 3 until the object module(s) are correctly built.

NOTE

The Makefile provided for the MVIDemo Protocol Handler uses different make rules for compiling a module with or without debug information. If debug information is to be included in the object module, fill in "all.d" in the Build Options field in the Program Builder. This make rule also defines a macro, `DEBUG`, which can be used in the source code for conditional compilation. If debug information is not to be included fill in "all" in the Build Options field in the Program Builder, or leave it empty. By convention, an object module with debug information is named "module name".do and an object module without debug information is named "module name".o.

NOTE

The dot at the end of the above commands denotes the current directory.

3.5.1.3 Download to Target

Assuming that a MVIBase development system is generated and installed as described in [Section 3.5.1.5, Build a Prommed System](#) and that the SLIP communication link between the HP 9000/700 work station and the CI535 submodule is set up and working, then take the following steps.

1. Change current directory to the test directory:

```
$ cd /ipa/projects/"UserProtocol"/members/  
"developer"/"MVIUser"/work/ MVIUserTest
```
2. Make sure that soft links are set up pointing to the object modules in the developer's working directory, see [Section 3.3.5, Test Directory](#). Note that it is not necessary to use soft links. Instead, you can copy the object modules from the working structure each time the modules are rebuilt.
3. Log in to the target system:

```
$ rlogin slip_target
```
4. You are now logged in to the target system and the VxWorks shell prompt is visible:

```
->
```
5. Set up the developer's login id and change the current directory to the test directory in the work station:

```
-> iam "developer"  
-> cd "slip_host:/ipa/projects/"UserProtocol"/members/  
"developer"/"MVIUser"/ work/MVIUserTest"
```
6. You are now ready to download the object module(s) into the target system. Use the VxWorks linking loader to do this:

```
-> ld < "protocol module".do
```
7. Start MVI Runtime Environment:

```
-> taskSpawn("tMVISup01", 110, 0, 1500, mviStartupMain, 0)
```

3.5.1.4 Debug in Target

Assuming that the Protocol Handler is to be debugged using the VxGDB remote debugger and that the modules are downloaded as described in [Section 3.5.1.3, Download to Target](#), take the following steps.

1. Start the debugger from a shell in the HP 9000/700 work station:

```
$ vxgdb &
```

2. From the debugger, connect to the target, set up directories indicating where to find the source code, set the initial break point and start the Protocol Handler:

```
(vxgdb &) target vxworks slip_target
(vxgdb &) dir
/ipa/projects/"UserProtocol"/members/"developer"/"MVIUser"/work/
ProtocolHandler
(vxgdb &) break protHndlMain
(vxgdb &) run protHndlMain 1
```

NOTE

You could do this using a start-up file as described in [Section 3.3.7, Debugger Initialization](#).

3. You are now ready to debug the Protocol Handler using VxGDB commands and GUI.

NOTE

The normal start-up procedure is that the start-up function, after creation of devices, spawns the Host Interface, which in turn spawns the Protocol Handler. You must temporarily disable this start-up procedure during the debugging phase. The reason for this is that the task to be debugged must be spawned under control of VxGDB. This is the reason why the development system includes the object module mviStartup.do, which do not spawn the Host Interface, instead of mviStartup.o which do spawn the Host Interface.

3.5.1.5 Build a Prommed System

You can build two kinds of MVIBase systems:

- A development system which includes support for networking, SLIP and remote debugging.
- A production system which excludes networking, SLIP and remote debugging, but includes the protocol developed.

Build a Development System

Assuming that MVIBase is installed as described in [Section 2.2.2, Software](#) and that preparations for system building are made as described in [Section 3.3.8, Preparation for System Building](#), take the following steps to build a development system.

1. Change current directory to the Integration subsystem directory of the development project:

```
$ cd /ipa/projects/"UserProtocol"/members/"developer"/
"MVIUser"/ work/Integration
```

2. Before a MVIBase system is built, you must specify the Internet addresses for the host (HP 9000/700 work station) and the target (CI535 submodule). See [Section 3.3.3, Host/Target Internet Addresses](#) and the description of the file inet.c above for details.

3. Build the system using the command:

```
$ mkTestSystem
```

This command generates a non-relocatable load module vwci535od.st and an s-record file named vwci535od.st.hex.

4. The s-record file, vwci535od.st.hex, is the file to be downloaded to the Flash PROM programmer and prommed.
5. Install the four Flash PROMs on the CI536 submodule as shown in [Figure D-2](#).
6. Install the submodule into the Advant Controller, and it is ready for use.

Build a Production System

Assuming that MVIBase is installed as described in [Section 2.2.2, Software](#) and that preparations for system building are made as described in [Section 3.3.8, Preparation for System Building](#), take the following steps to build a production system.

1. Change current directory to the source directory of the MVIBase product:

```
$ cd /ipa/projects/"UserProtocol"/members/"developer"/  
"MVIUser"/ work/Integration
```

2. Before a MVIBase system is built, you must specify the Internet addresses for the host system (HP 9000/700 work station) and the target system (CI535 submodule). See [Section 3.3.3, Host/Target Internet Addresses](#) and the description of the file inet.c above for details.

3. Build the system using the command:

```
$ mkProdSystem
```

This command generates a non-relocatable load module vwci535o.st and an s-record file named vwci535o.st.hex, which includes the MVIBase system as well as the protocol developed.

4. The s-record file, vwci535o.st.hex, is the file to be downloaded to the Flash PROM programmer and prommed.
5. Install the two Flash PROMs into the CI535 submodule as shown in [Figure D-1](#).
6. Install the submodule into the Advant Controller, and it is ready to use.

3.5.2 Demo Protocol Application Building for Advant Controller 400

This is a summary of the steps necessary to build the application data base and PC program for the MVI Demo Protocol in Advant Controller 400. See [Appendix B, MVI Demo Protocol Configuration](#) for a detailed description of the MVI Demo Protocol application.

1. Consider the following:
 - Network structure
 - Line characteristics
 - Function codes to be used for the data transfer
 - Register Addresses to be used
 - Cycle time for PC program.
2. Build the Configuration MS:
 - Line Characteristics MS (always necessary)
 - Network Configuration MS (always necessary)
 - Register Address MS (at least one MS necessary)
 - RTU Status MS (always necessary).
3. Build Command MS:

The Command MS are only used in master mode. For each slave, one Command MS is used for each function code.

 - Read Command MS
 - Write Command MS.
4. Build MS for data transfer:
 - Sending Data MS
 - Receiving Data MS.

NOTE

In order to update only part of a MS, there must always be a sending MS (IDENT = 1-96) corresponding to a receiving MS (IDENT = 101-196). The sending and receiving Data MS must be connected to the same DAT elements.

5. Build PC program for flow control:
 - The PC program is only necessary in master mode.

PC program

Use the PC program to control the Command MS if the Demo Protocol is used in master mode. The PC program is not necessary in slave mode.

The PC program controls and checks the data flow through the asynchronous port by means of the PC element SENDREQ and status information received from the CI535 submodule. The SENDREQ element activates the Command MS and the Data MS (see below). All SENDREQ PC elements used for Command MS to the same RTU must be included in the same control module (CONTRM). The PC program is described in [Section 3.6.7.4, Type Circuits for MVI Protocol Data Flow Control](#).

Data Base

The necessary data base for the CI535 submodule is defined in MVI Data Sets (MS). The MS data base is divided into three parts.

- **Configuration MS**

The configuration MS **must always be defined**. If they do not exist when you start up the CI535 submodule, application errors are reported.

- **Line Characteristics MS:** MVI Demo Protocol line characteristics (i.e., transmission speed, time-outs, etc.), one for each communication port. For a description of the MS, see [Section 3.6.7.1, MS for MVI Protocol Configuration](#).
- **Network Configuration MS:** Defines node numbers for all RTU nodes connected to the MVI Demo Protocol network. One MS is used for each communication port. For a description of the Network Configuration MS, see [Section 3.6.7.1, MS for MVI Protocol Configuration](#).
- **Register Address MS:** Holds the information for the cross-reference table on the CI535 submodule. The cross-reference table is used to translate the MVI Demo Protocol register addresses to corresponding Data MS identities in the Advant Controller 400.
One to four MS must be built for each port. For instructions on how to fill in the cross-reference table, see [Section 3.6.7.1, MS for MVI Protocol Configuration](#).
- **Status MS:** Receive status information for flow control and error indication used by the PC program and updated by the Demo Protocol. One Status MS is used for each port. For a description of the Status MS, see [Section 3.6.7.1, MS for MVI Protocol Configuration](#).

- **Command MS**

MS used for various commands to control slave nodes. The Command MS are only defined in the MVI Demo Protocol master node. See [Section 3.6.7.2, Command MS used in Master Mode](#).

- **Data MS**

MS used for the actual data transfer between the PC program and the RTU. There are two types of Data MS: Sending MS and Receiving MS. See [Section 3.6.7.3, Data MS for Data Transfer](#).

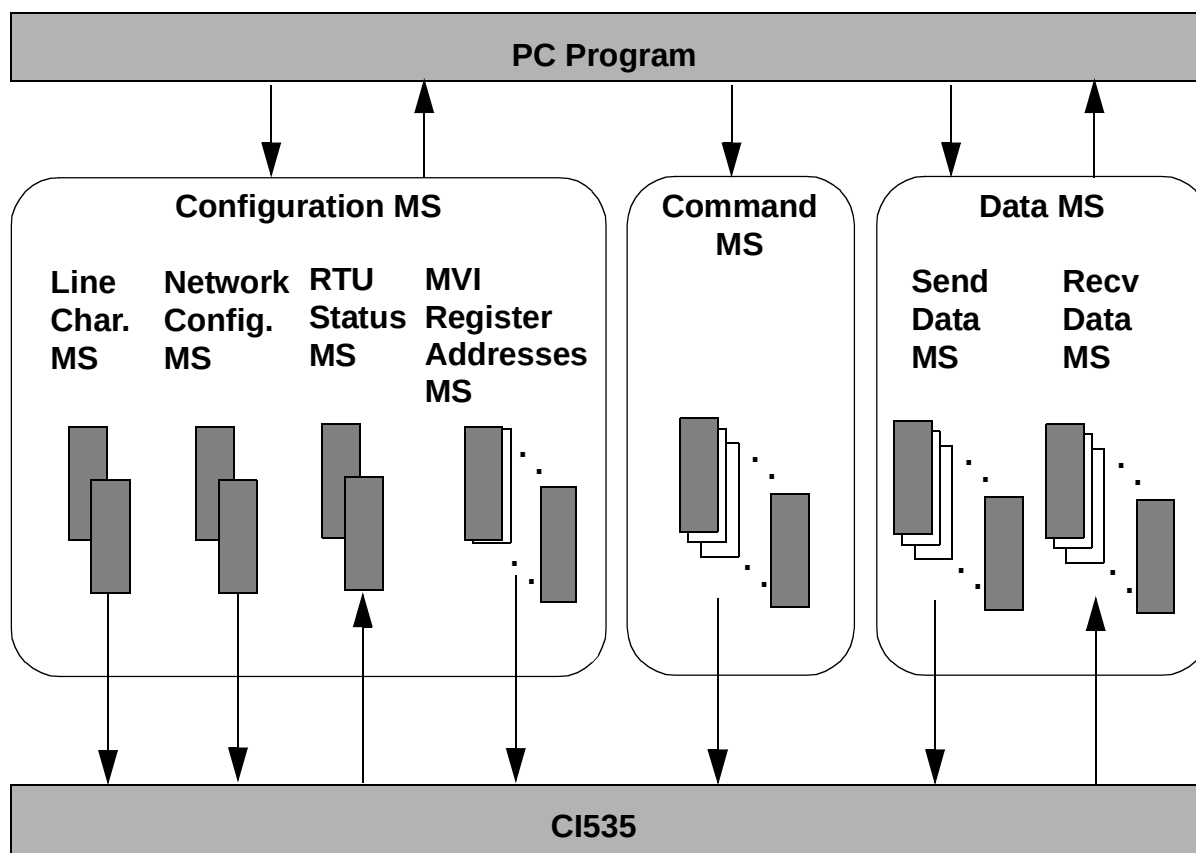


Figure 3-10. Overview Layout of the Application for MVI Demo Protocol

3.5.2.1 Master Functions

The functions mentioned below can be defined/controlled at **application** level in master mode:

- **Read registers (Function 1)**
Read Command MS (IDENT=202)
- **Write registers (Function 2)**
Write Command MS (IDENT=222).

See [Section 3.6.7.2, Command MS used in Master Mode](#) for a description of the Write Command MS and of the Read Command MS.

3.5.2.2 Slave Functions

It is not necessary to define any functions at application level in slave mode. Only the Configuration MS described in [Section 3.6.7.1, MS for MVI Protocol Configuration](#) and the Data MS for sending and receiving data must be defined.

3.6 Configuration/Application Building Menus

This section describes the function libraries comprising the MVI Runtime Environment. It also describes the MVI Demo Protocol application.

3.6.1 MVI UART Interface

The UART Interface is implemented as a character device driver as defined by the VxWorks I/O System. It provides functionality to control the device and handle the data flow to and from the device.

The Protocol Handler manipulates the UART by applying the system calls open, read, write, ioctl, select and close on the device name, which represents the instance of the UART in question. The VxWorks kernel, in turn, calls the corresponding routine in the device driver through the VxWorks I/O System.

For a description of data structures other than Read Descriptor, Write Descriptor and UART interrupt reasons, refer to the include files mviUARTLib.h and mviDRVLib.h, which are part of MVIBase.

3.6.1.1 Read Descriptor and Write Descriptor Data Structures

The Read Descriptor and Write Descriptor data structures are used, together with the data buffers, to hold information and data to be transferred between the Protocol Handler and the UART device driver.

MVI Read Descriptor

The Protocol Handler initiates a Read Descriptor with information about trigger mode, a pointer to the receive buffer, etc., and sends it to the UART device driver by means of a mviReqData call. See the fields of the Read Descriptor in [Figure 3-11](#) below.

```
typedef enum          /* MVI Read Operation Mode */
{
    MviRegard = 1 << 0,          /* Regard characters before CC's */
    MviSpecCC1 = 1 << 1,         /* Padding of CC 1 */
    MviORCondition = 1 << 2,      /* OR condition between the CC's */
    MviIdle = 1 << 7             /* Ignore all characters in the receive buffer */
} MviReadOperMode;

struct MviReadDesc
{
    1) MviReadOperMode modeOfOperation; /* Mode of Read operation */
    2) u_int16 maxBufLength;             /* Max buffer length */
    3) u_int16 postTrigLength;           /* Post trig condition */
    4) u_int16 noOfReceivedChars;        /* No. of received characters. */
    5) u_int8 theReceivedCC;             /* The received CC */
    6) u_int8 *readBuffer;               /* Read Buffer pointer */
    7) u_int8 CCOpt;                     /* CC option */
    8) u_int8 noOfCC;                    /* No. of CC */
    9) u_int8 CCSequence[6];            /* The CC list */
}
```

Figure 3-11. MVI Read Descriptor Data Structure

The fields in the MVI Read Descriptor are described below.

1. Five modes of operation can be defined:
 - a. modeOfOperation set to 0 (zero), which means normal operation, that is, all received characters are put in the buffer, except for default trigg sequence if specified.
 - b. MviRegard, which means that all characters received before the trigger characters specified in CCSequence are put in the read buffer.
 - c. MviSpecCC1, which means that padding of the first trigger character is enabled. When sending a message, each representation of the first trigger character is padded¹⁾ with another one. When a message is received, and the trigger condition is fulfilled, a check is made to see if an odd number of the first trigger character is received. If it is, the trigger condition is fulfilled. The padding characters are removed²⁾ from the received message. This is a way to detect the end of a message represented by a combination of two characters, even if the combination may occur within the message. This mode implies that noOfCC is set to 2.
 - d. MviORCondition, which means that there is an OR condition between the trigger characters specified, that is, any occurrence of one of the trigger characters in the message will fulfill the trigger condition. By default, there is an AND condition between the trigger characters specified in that order.
 - e. MviIdle, which means that the Read Descriptor does not specify a trigger condition for reception of a message. It is just a way of fetching possible events, such as character timeouts, etc., from the UART. All characters placed in the receive buffer before the event occurred are discarded.
2. Maximum length of the receive buffer. Maximum length is used by the UART device driver when the data buffer is filled.
3. This is the number of characters to receive after the CC sequence. Note that the specified length of the receive buffer must be large enough to hold both the CC:s and the number of characters specified. Relevant only if triggering on message length.
4. Number of characters put in the receive buffer, including the CC sequence. This field is updated by the UART Interface.
5. Updated with the received CC only if there is an OR condition between CC:s in the trigger condition. This field is updated by the UART Interface.
6. The pointer to the receive buffer.
7. The CC Option indicates whether the CC:s should be put in the receive buffer. A zero indicates rejection of the CC:s and a one indicates that the CC:s are put in the buffer.
8. This is the number of CC:s in the CC sequence.
9. The CC sequence, consisting of up to six CC:s.

1. The MVI Checksum Library contains a function, mviAddControl, for padding of control characters in a buffer. See [Section 3.6.3.7, MviAddControl](#).

2. The MVI Checksum Library contains a function, mviSkipControl, for removing control characters in a buffer. See [Section 3.6.3.6, MviSkipControl](#).

MVI Write Descriptor

The Protocol Handler initiates a Write Descriptor with buffer length and a pointer to the transmit buffer and sends it to the UART device driver by means of a write() system service. See the content of the Write Descriptor in [Figure 3-12](#) below.

```
struct MviWriteDesc /* MVI Write descriptor */
{
    u_int16 noOfChar; /* No. of characters to be sent */
    u_int8 *writeBuffer; /* Pointer to the transmit buffer */
};
```

Figure 3-12. MVI Write Descriptor Data Structure

3.6.1.2 UART Interrupt reasons

The interrupt reason bit field reported from UART Interface contains information about the reason for interrupt. It also contains status and error information about the reception/transmission and corresponding status/change status for the modem signals. See [Figure 3-13](#). This bit field is fetched from UART Interface by means of the ioctl commands

MviGetTXWakeupReason and **MviGetRXWakeupReason**.

```
typedef enum /* MVI Interrupt reason */
{
19) MviOverrunError = 1 << 0, /* Overrun error */
18) MviParityError = 1 << 1, /* Parity error */
17) MviFrameError = 1 << 2, /* Framing errpr */
16) MviCharTimeout = 1 << 3, /* Character timeout */
15) MviBreakInterrupt = 1 << 4, /* Break interrupt */
14) MviDSR = 1 << 5, /* Data Set Ready interrupt */
13) MviCTS = 1 << 6, /* Clear To Send interrupt */
12) MviDCD = 1 << 7, /* Data carrier Detect */
11) MviRI = 1 << 8, /* Ring Indicator interrupt */
10) MviCurrDSR = 1 << 9, /* DSR current status */
9) MviCurrCTS = 1 << 10, /* CTS current status */
8) MviCurrDCD = 1 << 11, /* DCD curent status */
7) MviCurrRI = 1 << 12, /* RI current status */
6) MviBufferOverrun = 1 << 14, /* Buffer overrun */
5) MviTimerInterrupt = 1 << 23 /* Timer interrupt */
4) MviRXFailed = 1 << 27, /* Reception failed */
3) MviRXCompleted = 1 << 28, /* Reception completed */
2) MviTXFailed = 1 << 30, /* Transmission failed */
1) MviTXCompleted = 1 << 31 /* Transmission completed */
} MviWakeupReason;
```

Figure 3-13. UART Interrupt Reasons

The interrupt reason bit field can be read from the Protocol Handler by means of an `ioctl()` function call. The bits in the UART Interrupt Reason bit field are described below.

1. TXCompleted: the transmission was completed successfully
2. TXFailed: the transmission failed
3. RXCompleted: the reception was completed successfully
4. RXFailed: the reception failed¹
5. TimerInterrupt: timeout time has expired
6. BufferOverflow: reception buffer overflowed. Buffer too small or speed too high.
7. CurrRI: current value of RI (reception)
8. CurrDCD: current value of DCD (reception)
9. CurrCTS: current value of CTS (transmission)
10. CurrDSR: current value of DSR (reception)
11. RI: status change of RI (reception)
12. DCD: status change of DCD (reception)
13. CTS: status change of CTS (transmission)
14. DSR: status change of DSR (reception)
15. BreakInterrupt: a break sequence received
16. CharacterTimeout: character timeout has occurred in reception
17. FramingError: a framing error has occurred in reception
18. ParityError: a parity error has occurred in reception
19. OverrunError: an overrun error has occurred in reception.

NOTE

RXFailed may be given as wakeup reason, even if the ModeOfOperation in the Read Descriptor is set to `MviIdle`. Consider this during implementation.

3.6.1.3 Open

NAME

`open` - system call

SYNOPSIS

```
int open(char* path, int oflag)
```

`path`, a pointer to a path name naming a file

`oflag`, a flag field describing the mode in which the file is opened. The value is constructed by OR-ing flags described below:

`O_RDONLY` open for reading only

`O_WRONLY` open for writing only

`O_RDWR` open for reading and writing

`O_NDELAY` this flag might affect subsequent reads and writes.

1. The reason for reception failure is given by the bits 6 and 16-19.

The files to be included are:

- ioLib.h
- mviDrvLib.h

DESCRIPTION

The open function initializes a UART device instance.

RETURNS

Upon successful completion, the file descriptor is returned. ERROR is returned if a file name is not specified, if the device does not exist, if no file descriptor is available, or if the driver returns ERROR. If the call fails, errno is set to indicate the error:

MviEMaxNoDevOpened - max number of devices already opened (Errno: 0x27100006)

MviEVhiCreateFailed - failed to create an instance of the Primitive Controller
(Errno: 0x27100007).

MviEVhiAttachFailed - failed to attach the Primitive Controller to its device
(Errno: 0x27100008).

3.6.1.4 Close

NAME

close - system call

SYNOPSIS

int close(int filedes)

filedes, a file descriptor obtained from an open system call

The files to be included are:

- ioLib.h
- mviDrvLib.h

DESCRIPTION

The close function releases a UART device instance.

RETURNS

The value returned by the driver or ERROR if the file descriptor is invalid. If the call fails, errno is set to indicate the error:

MviEVhiDetachFailed - failed to detach the Primitive Controller from its device
(Errno: 0x27100009).

MviEVhiDeleteFailed - failed to delete the instance of the Primitive controller
(Errno: 0x2710000a).

3.6.1.5 Read

The read system call is not used directly in MVI. Instead, use the mviReqData function, described in [Section 3.6.4.7, MviReqData](#), to start a read transaction, and use the mviGetData function, described in [Section 3.6.4.8, MviGetData](#), to fetch data from the UART.

3.6.1.6 Write

The write system call is not used directly in MVI. Instead, use the mviSendData function described in [Section 3.6.4.9, MviSendData](#).

3.6.1.7 ioctl

NAME

ioctl - system call

SYNOPSIS

```
int ioctl(int filedes, int request, int* arg)
```

filedes, a file descriptor obtained from an open system call

request, the particular command to perform.

arg, a pointer to an argument associated with the request

The files to be included are:

- ioLib.h
- mviDrvLib.h
- mviUARTLib.h

DESCRIPTION

The ioctl function is used to perform control functions on the UART device.

RETURNS

ERROR is returned if the file descriptor does not exist or an error occurs. If the call fails, errno is set to indicate the error:

MviEIllegalCommand - an unsupported command is requested (Errno: 0x2710000e)

MviEParamSetFailed - failed to set the value of a parameter (Errno: 0x27100000)

MviEParamGetFailed - failed to fetch the value of a parameter (Errno: 0x27100001)

MviEInitFailed - failed to initiate the SCC (Errno: 0x27100002)

MviEAssertDTRFailed - failed to assert DTR (Errno: 0x27100003)

MviEDeassertDTRFailed - failed to deassert DTR (Errno: 0x27100004).

The ioctl() function supports a number of commands as described in [Table 3-1](#).

Table 3-1. Ioctl Commands

Command	Argument ⁽¹⁾	Function Performed
MviCSetParameters	UART Parameter Block Pointer	Sets all parameters defined in the parameter block
MviCGetParameters	UART Parameter Block Pointer	Fetches all parameters defined in the parameter block

Table 3-1. Ioctl Commands (Continued)

Command	Argument ⁽¹⁾	Function Performed
MviCSetDuplex	Duplex: - MviFDuplex - MviHDuplex - MviHDuplexNoDCD	Sets Duplex mode
MviCSetPreambleTime	Number of character times	Sets Preamble time
MviCSetPostambleTime	Number of character times	Sets Postamble time
MviCSetXonXoffHandl	DxTrue = handle XON/XOFF DxFalse = don't handle XON/XOFF	Sets XON/XOFF handling mode
MviCSetBreakHandling	DxTrue = send break before transmission DxFalse = Don't send break before transmission	Sets Break Handling
MviCSetHalfDuplTmo	Number of character times	Sets Half Duplex timeout time
MviCSetBaudrate	Baudrate: - MviB150 - MviB300 - MviB600 - MviB1200 - MviB2400 - MviB4800 - MviB9600 - MviB19200 - MviB38400	Sets Baudrate
MviCSetParity	Parity Mode: - MviNoParity - MviOddParity - MviEvenParity - MviSpaceParity ⁽²⁾ - MviMarkParity ⁽³⁾	Sets Parity mode
MviCSetDataBits	7-bit or 8-bit code: - MviSevenDataBits - MviEightDataBits	Sets number of Databits in a character
MviCSetStopBits	Number of Stop bits: - MviHalfStopBit - MviOneStopBit - MviOneAndHalfStopBit - MviTwoStopBits	Sets number of Stop bits in a character
MviCSetCharTimeout	Number of character times	Sets the Character Timeout time
MviCGetDuplex	A pointer to the variable to which to return the value	Fetches the Duplex mode previously set

Table 3-1. Ioctl Commands (Continued)

Command	Argument ⁽¹⁾	Function Performed
MviCGetPreambleTime	A pointer to the variable to which to return the value	Fetches the Preamble time previously set
MviCGetPostambleTime	A pointer to the variable to which to return the value	Fetches the Postamble time previously set
MviCGetXonXoffHandl	A pointer to the variable to which to return the value	Fetches the XON/XOFF handling mode previously set
MviCGetBreakHandling	A pointer to the variable to which to return the value	Fetches the Break handling mode previously set
MviCGetHalfDuplTmo	A pointer to the variable to which to return the value	Fetches the Half Duplex Timeout time previously set
MviCGetBaudrate	A pointer to the variable to which to return the value	Fetches the Baudrate previously set
MviCGetParity	A pointer to the variable to which to return the value	Fetches the Parity mode previously set
MviCGetDataBits	A pointer to the variable to which to return the value	Fetches the Data Bits code previously set
MviCGetStopBits	A pointer to the variable to which to return the value	Fetches the number of Stop bits previously set
MviCGetCharTimeout	A pointer to the variable to which to return the value	Fetches the Character Timeout time previously set
MviCInitUART	UART Init Descriptor pointer	Initiates the UART
MviCAssertDTR	NULL	Asserts the Data Terminal Ready modem signal
MviCDeassertDTR	NULL	Deasserts the Data Terminal Ready modem signal
MviCGetRxWakeupReason	UART Interrupt Reason pointer	Fetches the reason for waking up in select when waiting for read completion
MviCGetTxWakeupReason	UART Interrupt Reason pointer	Fetches the reason for waking up in select when waiting for write completion

(1) The Argument is of type pointer to integer.

(2) MviSpaceParity is not implemented.

(3) MviMarkParity is not implemented.

3.6.1.8 Select

NAME

select - system call

SYNOPSIS

```
struct timeval { long tv_sec; long tv_usec; }
```

```
int select(int nfds, int* readfds, int* writefds, int* exceptfds, struct timeval* timeout)
```

nfds, the highest numbered file descriptor that should be checked plus 1

readfds, a bitmask which represents read file descriptors that should be checked.

The bitmask is defined according to the formula $(1 \ll fd1) + (1 \ll fd2) + \dots$

writefds, a bitmask which represents write file descriptors that should be checked.

The bitmask is defined according to the formula $(1 \ll fd1) + (1 \ll fd2) + \dots$

exceptfds, a bitmask which represents exception file descriptors that should be checked.

The bitmask is defined according to the formula $(1 \ll fd1) + (1 \ll fd2) + \dots$

timeout, a pointer to a timeval struct specifying a timeout value for select(). This value is used by the select() system call to return after a certain amount of time, even if I/O has not been performed

The files to be included are:

- ioLib.h
- selectLib.h
- mviDrvLib.h

DESCRIPTION

The select() system call allows a process to check if one or more devices is capable of doing I/O. If none of the requested devices are ready, the system blocks the process until one (or several) becomes ready.

RETURNS

Upon successful completion, the number of file descriptors ready for I/O is returned, and the bitmasks are modified to indicate which file descriptor(s) are ready. If the timeout time expires, select() returns OK and all masks are cleared. Otherwise select() returns ERROR and errno contains the reason for the error.

EXAMPLE

```
void example_of_select(void)
{
    .....
    int width;
    fd_set readFds, writeFds,
    int fd1, fd2;
    MviReadDesc readDesc;
    MviWriteDesc writeDesc
    MviWakeupReason wakeupReason;
    .....
    fd1 = open(...); fd2 = open(...);
    width = (fd1 > fd2) ? fd1 : fd2; width++;
    .....
    /* Initialize the read descriptor with trig condition */
    .....
    mviReqData( fd1, &readDesc );
    FD_ZERO( &readFds ); FD_SET( fd1, &readFds );
    select(width, &readFds, NULL, NULL, NULL);
    if (FD_ISSET(fd1,&readFds))
    {
        ioctl(fd1, MviCGetRxWakeupReason, wakeupReason);
        if (wakeupReason & MviRXCompleted)
        {
            /* mviSendData() succeeded */
            mviGetData(fd1, &readDesc)
            writeDesc.writeBuffer = readDesc.readBuffer;
            writeDesc.noOfChar = readDesc.noOfReceivedChars;
            mviSendData( fd2, &writeDesc);
            FD_ZERO( &writeFds ); FD_SET( fd2, &writeFds );
            select(width, NULL, &writeFds, NULL, NULL);
            if (FD_ISSET(fd2,&writeFds))
            {
                ioctl(fd2, MviCGetTxWakeupReason, wakeupReason);
                if (wakeupReason & MviTXCompleted)
                {
                    /* mviSendData() succeeded */
                }
                else
                {
                    /* mviSendData() failed */
                }
            }
        }
    }
    else
    {
        /* mviReqData() failed */
    }
}
}
```

3.6.2 MVI Watchdog Timers

The MVI Watchdog Timer function is implemented as a character device driver as defined by the VxWorks I/O System. It provides a set of functions to control the device. This device driver encapsulates the VxWorks Watchdog Timer function, making it possible to wait for a Watchdog Timer to expire in select.

A Watchdog Timer can be opened using the open() system call, closed using the close() system call, waited for using the select() system call and manipulated using the ioctl() system call. The ioctl commands supported are as follows.

- MviCWDTStart: start the Watchdog Timer. The ioctl argument should contain the delay time in number of milliseconds.
- MviCWDTCancel: cancel a running Watchdog Timer. The ioctl argument is ignored by this command.

Since the VxWorks implementation of select does not support the exception mask, the read mask is used instead. To be able to wait in select for a Watchdog Timer to expire, the timer device file descriptor should be set in the read mask of the select call.

3.6.2.1 Open

NAME

open - system call

SYNOPSIS

```
int open(char* path, int oflag)
```

path, a pointer to a path name naming a file

oflag, a flag field describing the mode in which the file is opened. The value is constructed by OR-ing flags described below:

O_RDONLY open for reading only

O_WRONLY open for writing only, O_RDWR open for reading and writing

O_NDELAY this flag might affect subsequent reads and writes

The files to be included are:

- ioLib.h

- mviDrvLib.h

DESCRIPTION

The open function initializes a Watchdog Timer device instance.

RETURNS

Upon successful completion, the file descriptor is returned. ERROR is returned if a file name is not specified, if the device does not exist, if no file descriptors are available or if the driver returns ERROR. If the call fails, errno is set to indicate the error:

MviEMaxNoDevOpened - max number of devices already opened (Errno: 0x27100006)

MviEWDCreateFailed - creating a Watchdog Timer failed.

3.6.2.2 Close

NAME

close - system call

SYNOPSIS

int close(int filedes)

filedes, a file descriptor obtained from an open system call

The files to be included are:

- ioLib.h
- mviDrvLib.h

DESCRIPTION

The close function releases a Watchdog Timer device instance.

RETURNS

The value returned by the driver, or ERROR if the file descriptor is invalid. If the call fails, errno is set to indicate the error:

MviEWDDDeleteFailed - failed to delete the Watchdog Timer.

3.6.2.3 ioctl

NAME

ioctl - system call

SYNOPSIS

int ioctl(int filedes, int request, int* arg)

filedes, a file descriptor obtained from an open system call

request, the particular command to perform.

arg, a pointer to an argument associated with the request

The files to be included are:

- ioLib.h
- mviDrvLib.h

DESCRIPTION

The ioctl function is used to perform control functions on the Watchdog Timer device.

The functions supported are:

- MviCWDTStart. Start the Watchdog Timer. The ioctl argument should contain the delay time in number of milliseconds.
- MviCWDTCancel. Cancel a running Watchdog Timer. The ioctl argument is ignored by this command.

RETURNS

ERROR is returned if the file descriptor does not exist or an error has occurred. If the call fails, errno is set to indicate the error:

MviIllegalCommand - an unsupported command is requested (Errno: 0x2710000e).

3.6.2.4 Select

NAME

select - system call

SYNOPSIS

```
struct timeval { long tv_sec; long tv_usec; }
```

```
int select(int nfds, int* readfds, int* writefds, int* exceptfds, struct timeval* timeout)
```

nfds, the highest numbered file descriptor that should be checked plus 1

readfds, a bitmask which represents read file descriptors that should be checked.

The bitmask is defined according to the formula $(1 \ll fd1) + (1 \ll fd2) + \dots$

writefds, a bitmask which represents write file descriptors that should be checked. The bit mask is defined according to the formula $(1 \ll fd1) + (1 \ll fd2) + \dots$

exceptfds, a bitmask which represents exception file descriptors that should be checked.

The bitmask is defined according to the formula $(1 \ll fd1) + (1 \ll fd2) + \dots$

timeout, a pointer to a timeval struct specifying a timeout value for select(). This value is used by the select() system call to return after a certain amount of time, even if I/O has not been performed.

The files to be included are:

- ioLib.h
- selectLib.h
- mviDrvLib.h

DESCRIPTION

The select() system call allows a process to check if one or more devices are capable of doing I/O. If none of the requested devices are ready, the system blocks the process until one (or several) becomes ready.

RETURNS

Upon successful completion, the number of file descriptors ready for I/O is returned, and the bitmasks are modified to indicate which file descriptor(s) are ready. If the timeout time expires, select() returns OK and all masks are cleared. Otherwise, select() returns ERROR and errno contains the reason for the error.

EXAMPLE:

```
void example_of_select(void)
{
    .....
    int width;
    fd_set readFds, writeFds,
    int fd1, fd2, fd3;
    int wdTime = 100; /* Timeout time in milliseconds */
    MviReadDesc readDesc;
    MviWriteDesc writeDesc
```

```
MviWakeupReason wakeupReason;
.....
fd1 = open("mviUART/1, ..."); fd2 = open("mviUART/2", ...);
fd3 = open("/watchdogTimer/1",...);
width = (fd1 > fd2) ? fd1 : fd2; width++;
.....
/* Initialize the read descriptor with trig condition */
.....
mviReqData( fd1, &readDesc );
ioctl(fd3, MviCWDTStart, &wdTime);
FD_ZERO( &readFds ); FD_SET( fd1, &readFds );
FD_SET( fd3, &readFds );
select(width, &readFds, NULL, NULL, NULL);
if (FD_ISSET(fd1,&readFds))
{
    /* Event from UART */
    ioctl(fd1, MviCGetRxWakeupReason, wakeupReason);
    if (wakeupReason & MviRXCompleted)
    {
        /* mviSendData() succeeded */
        mviGetData(fd1, &readDesc)
        writeDesc.writeBuffer = readDesc.readBuffer;
        writeDesc.noOfChar = readDesc.noOfReceivedChars;
        mviSendData( fd2, &writeDesc);
        FD_ZERO( &writeFds ); FD_SET( fd2, &writeFds );
        select(width, NULL, &writeFds, NULL, NULL);
        if (FD_ISSET(fd2,&writeFds))
        {
            ioctl(fd3, MviCWDTCancel, 0);
            ioctl(fd2, MviCGetTxWakeupReason, wakeupReason);
            if (wakeupReason & MviTXCompleted)
            {
                /* mviSendData() succeeded */
            }
            else
            {
                /* mviSendData() failed */
            }
        }
    }
}
else
{
    /* mviReqData() failed */
}
}
else
{
    /* Event from watchdog Timer (Watchdog Timer expired) */
    .....
}
}
```

3.6.3 MVI Checksum Library

This section describes functions for checksum calculations and related services supported by the MVI Runtime Environment. The functions supported are as follows.

- mviCrc_16: CRC-16 checksum calculation.
- mviBcc: BCC checksum calculation.
- mviLrc: LRC checksum calculation.
- mviCrc_8: CRC-8 checksum calculation.
- mviSum: Arithmetic sum checksum calculation.
- mviSkipControl: Takes away control characters from a buffer.
- mviAddControl: Adds a control character after another in a buffer.

3.6.3.1 MviCrc_16

NAME

mviCrc_16 - calculates the CRC-16 value for a data buffer

SYNOPSIS

u_int16 mviCrc_16 (u_int8 buffer, int length, u_int16 poly, int version, u_int16 accSet)

buffer, a pointer to the buffer where the message is stored for which the checksum is to be calculated

length, the length of the message (bytes)

poly, the crc-polynom

version, calculation method (1 = normal, 2 = reversed)

accSet, the value that the accumulator will get before the crc calculation starts

The files to be included are:

- mviChkSumLib.h

DESCRIPTION

The function calculates the CRC-16 value for a data buffer. The calculation method used is either normal or reversed, depending on the version parameter. The polynom used in the calculation is chosen by the user with the poly parameter.

NOTE

When you are using the reversed calculation method, you must byte-swap the returned CRC-value before it is transmitted.

EXAMPLE

An example of how the CRC-16 checksum is calculated using version=1 (reversed calculation) and 0xffff as accumulator preset follows. The polynomial used is 10100000000000101. After the accumulator is loaded with the accumulator preset, the step-by-step procedure to form the CRC-value is as follows:

1. Shift the data byte and the accumulator one step to the right.
- 2a. If the bit shifted out from the data byte exclusive-OR'ed with the bit shifted out from the accumulator is 1 (one), then the accumulator should be exclusive-OR'ed with the polynomial.
- 2b. If the bit shifted out from the data byte exclusive-OR'ed with the bit shifted out from the accumulator is 0 (zero), then return to step 1.
3. Repeat 1 and 2 until eight shifts have been performed.
4. Fetch new data byte.
5. Repeat steps 1 - 4 until all bytes of the message have been used.

Message = {0x02, 0x07}

	Data	0000 0010	(0x02)
	Acc	1111 1111 1111 1111	
Shift 1	Data	0000 0001	-> 0
	Acc	0111 1111 1111 1111	-> 1
	Poly	1010 0000 0000 0001	
		1101 1111 1111 1110	
Shift 2	Data	0000 0000	-> 1
	Acc	0110 1111 1111 1111	-> 0
	Poly	1010 0000 0000 0001	
		1100 1111 1111 1110	
Shift 3	Data	0000 0000	-> 0
	Acc	0110 0111 1111 1111	-> 0
Shift 4	Data	0000 0000	-> 0
	Acc	0011 0011 1111 1111	-> 1
	Poly	1010 0000 0000 0001	
		1001 0011 1111 1110	
Shift 5	Data	0000 0000	-> 0
	Acc	0100 1001 1111 1111	-> 0
Shift 6	Data	0000 0000	-> 0
	Acc	0010 0100 1111 1111	-> 1
	Poly	1010 0000 0000 0001	
		1000 0100 1111 1110	
Shift 7	Data	0000 0000	-> 0
	Acc	0100 0010 0111 1111	-> 0
Shift 8	Data	0000 0000	-> 0
	Acc	0010 0001 0011 1111	-> 1
	Poly	1010 0000 0000 0001	
		1000 0001 0011 1110	
	Data	0000 0111	(0x07)
	Acc	1000 0001 0011 1110	

Shift 1	Data		0000 0011	-> 1
	Acc	0100 0000	1001 1111	-> 0
	Poly	1010 0000	0000 0001	
		1110 0000	1001 1110	
Shift 2	Data		0000 0001	-> 1
	Acc	0111 0000	0100 1111	-> 0
	Poly	1010 0000	0000 0001	
		1101 0000	0100 1110	
Shift 3	Data		0000 0000	-> 1
	Acc	0110 1000	0010 0111	-> 0
	Poly	1010 0000	0000 0001	
		1100 1000	0010 0110	
Shift 4	Data		0000 0000	-> 0
	Acc	0110 0100	0001 0011	-> 0
Shift 5	Data		0000 0000	-> 0
	Acc	0011 0010	0000 1001	-> 1
	Poly	1010 0000	0000 0001	
		1001 0010	0000 1000	
Shift 6	Data		0000 0000	-> 0
	Acc	0100 1001	0000 0100	
Shift 7	Data		0000 0000	-> 0
	Acc	0010 0100	1000 0010	-> 0
Shift 8	Data		0000 0000	-> 0
	Acc	0001 0010	0100 0001	-> 0
		1 2 4 1		
Checksum = 0x1241				

Another example of how CRC-16 checksum is calculated, this time using version = 1 (normal calculation) and 0x0000 as accumulator preset, follows. The polynomial used is 1000000000000101. The step-by-step procedure to form the CRC-value is as follows:

1. Shift the data byte and the accumulator one step to the left.
- 2a. If the bit shifted out from the data byte exclusive-OR'ed with the bit shifted out from the accumulator is 1 (one), then the accumulator should be exclusive-OR'ed with the polynomial.
- 2b. If the bit shifted out from the data byte exclusive-OR'ed with the bit shifted out from the accumulator is 0 (zero), then return to step 1.
3. Repeat 1 and 2 until eight shifts have been performed.
4. Fetch new data byte.
5. Repeat steps 1 - 4 until all bytes of the message have been used.

Message = {0x02, 0x07}

	Data		0000 0010
Shift 1	Data	0 <-	0000 0100
	Acc	0 <-	0000 0000 0000 0000
Shift 2	Data	0 <-	0000 1000
	Acc	0 <-	0000 0000 0000 0000
Shift 3	Data	0 <-	0001 0000
	Acc	0 <-	0000 0000 0000 0000

```

Shift 4  Data      0 <-  0010 0000
        Acc       0 <-  0000 0000 0000 0000
Shift 5  Data      0 <-  0100 0000
        Acc       0 <-  0000 0000 0000 0000
Shift 6  Data      0 <-  1000 0000
        Acc       0 <-  0000 0000 0000 0000
Shift 7  Data      1 <-  0000 0000
        Acc       0 <-  0000 0000 0000 0000
        Poly      1000 0000 0000 0101
                1000 0000 0000 0101
Shift 8  Data      0 <-  0000 0000
        Acc       1 <-  0000 0000 0000 1010
        Poly      1000 0000 0000 0101
                1000 0000 0000 1111
        Data      0000 0111
        Acc       1000 0000 0000 1111
Shift 1  Data      0 <-  0000 1110
        Acc       1 <-  0000 0000 0001 1110
        Poly      1000 0000 0000 0101
                1000 0000 0001 1011
Shift 2  Data      0 <-  0001 1100
        Acc       1 <-  0000 0000 0011 0110
        Poly      1000 0000 0000 0101
                1000 0000 0011 0011
Shift 3  Data      0 <-  0011 1000
        Acc       1 <-  0000 0000 0110 0110
        Poly      1000 0000 0000 0101
                1000 0000 0110 0011
Shift 4  Data      0 <-  0111 0000
        Acc       1 <-  0000 0000 1100 0110
        Poly      1000 0000 0000 0101
                1000 0000 1100 0011
Shift 5  Data      0 <-  1110 0000
        Acc       1 <-  0000 0001 1000 0110
        Poly      1000 0000 0000 0101
                1000 0001 1000 0011
Shift 6  Data      1 <-  1100 0000
        Acc       1 <-  0000 0011 0000 0110
Shift 7  Data      1 <-  1000 0000
        Acc       0 <-  0000 0110 0000 1100
        Poly      1000 0000 0000 0101
                1000 0110 0000 1001
Shift 8  Data      1 <-  0000 0000
        Acc       1 <-  0000 1100 0001 0010
        Hex       0    c    1    2
Checksum = 0x0c12

```

RETURNS

The calculated CRC-value.

3.6.3.2 MviBcc

NAME

mviBcc - calculates the BCC checksum for a data buffer

SYNOPSIS

u_int8 mviBcc (u_int8 buffer, int length)

buffer, a pointer to the buffer where the message is stored for which the checksum is to be calculated

length, the length of the message (bytes)

The files to be included are:

- mviChkSumLib.h

DESCRIPTION

The function calculates the BCC checksum for a data buffer.

EXAMPLE

An example of how the BCC-value is calculated using {0x02, 0x07} as input follows.

The step-by-step procedure to form the BCC-checksum is as follows:

1. Exclusive OR the data byte with the accumulator.
2. Fetch new data byte.
3. Repeat 1 and 2 until all bytes of the message are used.

Data	0000	0010
Acc	0000	0000
	0000	0010
Data	0000	0111
Acc	0000	0010
	0000	0101
Hex	0	5
Checksum	= 0x05	

RETURNS

The calculated BCC-value.

3.6.3.3 MviLrc

NAME

mviLrc - calculates the LRC checksum for a data buffer

SYNOPSIS

u_int8 mviLrc (u_int8 buffer, int length)

buffer, a pointer to the buffer where the message is stored for which the checksum is to be calculated

length, the length of the message (bytes)

The files to be included are:

- mviChkSumLib.h

DESCRIPTION

The function calculates the LRC checksum for a data buffer.

EXAMPLE:

An example of how the LRC-value is calculated using {0x02, 0x07} as input follows. The step-by-step procedure to form the LRC-checksum is as follows.

1. Add data byte to accumulator without wraparound carry.
2. Fetch new data byte.
3. Repeat 1 and 2 until all bytes of the message have been used.
4. Invert accumulator and add 1.

Data	0000	0010
Acc	0000	0000
	0000	0010
Data	0000	0111
Acc	0000	0010
	0000	1001
Invert	1111	0110
Add	11111	0111
Hex	f	7
Checksum	= 0xf7	

RETURNS

The calculated LRC-value.

3.6.3.4 MviCrc_8

NAME

mviCrc_8 - calculates the CRC-8 value for a data buffer

SYNOPSIS

u_int8 mviCrc_8 (u_int8 buffer, int length, u_int8 poly, int version, u_int8 accSet)

buffer, a pointer to the buffer where the message is stored for which the checksum is to be calculated

length, the length of the message (bytes)

poly, the crc-polynom

version, calculation method (1 = normal, 2 = reversed)

accSet, the value that the accumulator gets before the CRC calculation starts.

The files to be included are:

- mviChkSumLib.h

DESCRIPTION

The function calculates the CRC-8 value for a data buffer. The calculation method used is either normal or reversed, depending on the version parameter. The polynom used in the calculation is chosen by the user with the poly parameter.

RETURNS

The calculated CRC-value.

3.6.3.5 MviSum

NAME

mviSum - calculates the arithmetic checksum for a data buffer

SYNOPSIS

u_int8 mviSum (u_int8 buffer, int length)

buffer, a pointer to the buffer where the message is stored for which the checksum is to be calculated

length, the length of the message (bytes)

The files to be included are:

- mviChkSumLib.h

DESCRIPTION

The function calculates the arithmetic checksum for a data buffer. The message bytes are added using no carry.

EXAMPLE

An example of how the SUM-value is calculated using {0x02, 0x07} as input follows.
The step-by-step procedure to form the arithmetic-checksum is as follows:

1. Add the data byte to the accumulator without wraparound carry.
2. Fetch new data byte.
3. Repeat 1 and 2 until all bytes of the message have been used.

Data	0000	0010
Acc	0000	0000
	0000	0010
Data	0000	0111
Acc	0000	0010
	0000	1001
Hex	0	9
Checksum	= 0x09	

RETURNS

The calculated SUM-value.

3.6.3.6 MviSkipControl

NAME

mviSkipControl - search for and delete all control characters or every other control character mentioned

SYNOPSIS

```
int mviSkipControl(u_int8 charToDelete, u_int8 bufferIn, int length,  
                  u_int8 bufferOut, int version)
```

charToDelete, control character to be deleted

bufferIn, a pointer to the buffer where the message is stored

length, the length of the message (bytes)

bufferOut, a pointer to the buffer where the modified message is placed

version, 1 = remove all char, 2 = remove every second char

The files to be included are:

- mviChkSumLib.h

DESCRIPTION

Search for and delete all control characters or every other control character mentioned.

RETURNS

The length of the modified message.

3.6.3.7 MviAddControl

NAME

mviAddControl - search for a control character and adds another

SYNOPSIS

```
int mviAddControl (u_int8 charToFind, uint_8 charToAdd, u_int8* bufferIn,  
                  int length, u_int8* bufferOut, int version)
```

charToFind, control character to search for

charToAdd, control character to add

bufferIn, a pointer to the buffer where the message is stored

length, the length of the message (bytes)

bufferOut, a pointer to the buffer where the modified message is placed

version 1 = Add the specified control character before the control character searched for

2 = Add the specified control character after the control character searched for

The files to be included are:

- mviChkSumLib.h

DESCRIPTION

Searches for a control character and adds another, after or before, the mentioned control character

RETURNS

The length of the modified message or MviBufSize if bufferOut is too small for modified message.

3.6.4 MVI Runtime Library

This section describes functions for allocation, initialization, manipulation and deallocation of MVI Runtime Environment data structures. The functions supported are:

- mviGetNodeDLEntry - Get the DL-entry for a given node
- mviGetNodeStatusInfo - Get status information for a given node
- mviGetNodeDiagInfo - Get diagnostic information for a given node
- mviInitUART - Initiate UART device
- mviCreateUARTDevice - Create UART device for the instance spawned
- mviOpenUARTDevice - Open UART device for the instance spawned
- mviReqData - Sends the Read Descriptor to the UART device
- mviGetData - Fetches data from the UART device
- mviSendData - Sends data through the UART device
- mviSelect - Waits on a number of file descriptors for an event to occur
- mviSendLinkStatus - Setting up or taking down a connection between the application and a RTU on the communication link

- mviCreatePHIDevices - Create Protocol Handler Interface devices
- mviCreateDLIDevices - Create Data Link Handler Interface devices
- mviOpenPHIDevices - Open Protocol Handler Interface devices
- mviOpenDLIDevices - Open Data Link Handler interface
- mviCreateTimerDevices - Create Watchdog Timer devices
- mviOpenTimerDevices - Open Watchdog Timer devices
- mviFetchSetupInfo - Fetch initialization information from application
- mviInitDLStruct - Initialize the DL structure
- mviInitGlobalDataStruct - Initializes the common part of the GlobalData structure
- mviMakeWDTDevName - Creates a device name string for Watchdog Timer devices
- mviCleanup - Deallocates data structures and closes devices opened
- mviSendDLSetupInfo - Send setup info
- mviFetchDLSetupInfo - Receive setup info
- mviWaitForModemSignals - Waits for modem signals to become active
- mviSysMess - Send a system message

For descriptions of data structures, refer to the include file mviDLLib.h, which is part of MVIBase.

3.6.4.1 MviGetNodeDLEntry

NAME

mviGetNodeDLEntry - get the DL-entry for a given node

SYNOPSIS

```
MviStatus mviGetNodeDLEntry(MviDL* dlPtr, u_int8 nodeNo,  
                             MviDLEntry **dlEntryPtr)
```

dlPtr, a pointer to the DL data structure

nodeNo, node number

dlEntryPtr, a pointer to a pointer to a MviDLEntry struct

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function searches the DL-struct for a DL-entry corresponding to a given node number. The pointer to the DL-entry found is then copied to the pointer to a MviDLEntry struct.

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.2 MviGetNodeStatusInfo

NAME

mviGetNodeStatusInfo - get status information for a given node

SYNOPSIS

```
MviStatus mviGetNodeStatusInfo(MviDL* dlPtr, u_int8 nodeNo,  
                               MviNodeStat **nodeStatPtr)
```

dlPtr, a pointer to the DL data structure

nodeNo, node number

nodeStatPtr, a pointer to a pointer to a MviNodeStat struct

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function searches the status information struct for a node status entry corresponding to a given node number. The pointer to the node status entry found is then copied to the pointer to a MviNodeStat struct.

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.3 MviGetNodeDiagInfo

NAME

mviGetNodeDiagInfo - get diagnostic information for a given node

SYNOPSIS

```
MviStatus mviGetNodeDiagInfo(MviDL* dlPtr, u_int8 nodeNo,  
                             MviNodeDiag **nodeDiagPtr)
```

dlPtr, a pointer to the DL data structure

nodeNo, node number

nodeDiagPtr, a pointer to a pointer to a MviNodeDiag struct

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function searches the diagnostic information struct for a node diagnostic entry corresponding to a given node number. The pointer to the node diagnostic entry found is then copied to the pointer to a MviNodeDiag struct.

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.4 MviInitUART

NAME

mviInitUART - initiate UART device

SYNOPSIS

```
MviStatus mviInitUART(int fileDesc,  
                      MviProtSetupInfo* setupDataPtr,  
                      MviUARTParBlk* parameterBlockPtr,  
                      MviInitDescriptor* initDescriptorPtr,  
                      MviHandleXonXoff xonXoff,  
                      MviSendBreak sendBreak,  
                      DxBoolean ringIndIrq,  
                      DxBoolean breakIrq)
```

fileDesc, file descriptor for the UART device

setupDataPtr, a pointer to the setup data structure

parameterBlockPtr, a pointer to the UART parameter data structure

initDescriptorPtr, a pointer to the UART init descriptor data structure

xonXoff defines whether xon/xoff is handled

- DxTrue = xon/xoff is handled

sendBreak defines whether a break character is sent before transmission

- DxTrue = break is sent before transmission

ringIndIrq defines whether an interrupt is generated upon a status change of RI

- DxTrue = interrupt is generated

breakIrq defines whether an interrupt is generated upon reception of a break character

- DxTrue = interrupt is generated

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function is used by the Protocol Handler to make a full initialization of the UART device. It assumes that the setupDataPtr is initialized.

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.5 MviCreateUARTDevice

NAME

mviCreateUARTDevice - create UART device for the instance spawned

SYNOPSIS

MviStatus mviCreateUARTDevice(int taskInstance)

taskInstance, instance number of the task spawned. Instances are numbered 1 - n.

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function creates the UART devices used in MVI.

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.6 MviOpenUARTDevice

NAME

mviOpenUARTDevice - open UART device for the instance spawned

SYNOPSIS

MviStatus mviOpenUARTDevice(int* uartFdPtr, int taskInstance)

uartFdPtr, a pointer to the UART file descriptor

taskInstance, instance number of the task spawned. Instances are numbered 1 - n.

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function opens the UART device which is to be used by the Protocol Handler instance.

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.7 MviReqData

NAME

mviReqData - sends the Read Descriptor to the UART device

SYNOPSIS

mviReqData(int fd, MviReadDesc *readDesc)

fd, file descriptor to device

readDesc, a pointer to the Read Descriptor which is to be used by the UART device.

The files to be included are:

- mviDLLib.h

DESCRIPTION

Sends the Read Descriptor to the UART device.

RETURNS

0 if successful, -1 if error.

3.6.4.8 MviGetData

NAME

mviGetData - fetches data from the UART device

SYNOPSIS

mviGetData(int fd, MviReadDesc *readDesc)

fd, file descriptor to device

readDesc, a pointer to the Read Descriptor.

The files to be included are:

- mviDLLib.h

DESCRIPTION

Gets the data from the UART device.

RETURNS

The number of received characters, -1 if error.

3.6.4.9 MviSendData

NAME

mviSendData - sends data through the UART device

SYNOPSIS

mviSendData(int fd, MviWriteDesc *writeDesc)

fd, file descriptor to device

writeDesc, a pointer to the Write Descriptor

The files to be included are:

- mviDLLib.h

DESCRIPTION

Sends the data through the UART device.

RETURNS

-1 if error.

3.6.4.10 MviSelect

NAME

mviSelect - waits on a number of file descriptors for an event to occur

SYNOPSIS

```
int mviSelect(int nfds,  
              MviFdSet* readMask,  
              MviFdSet* writeMask,  
              int watchdogToutFd,  
              int watchdogFd,  
              DxBoolean selectBlocked)  
  
nfds, number of file descriptors  
readMask, a pointer to the read mask or NULL  
writeMask, a pointer to the write mask or NULL  
watchdogToutFd, file descriptor for Watchdog timeout timer  
watchdogFd, file descriptor for Watchdog message  
selectBlocked, defines whether to wait in select until an event occurs  
-DxTrue = wait for event
```

The files to be included are:

- mviDLLib.h

DESCRIPTION

Besides making a select call, this function hides the sending of Watchdog messages to the Host System.

RETURNS

The number of file descriptors which are ready for I/O if the call succeeds, otherwise ERROR. If the call succeeds, the bitmasks are modified to indicate which file descriptors are ready. If the timeout time expires, OK is returned and all masks are cleared.

EXAMPLE

```
void example_of_select(void)
{
    .....
    int width;
    fd_set readFds, writeFds,
    int fd1, fd2, wdToutFd, wdFd;
    MviReadDesc readDesc;
    MviWriteDesc writeDesc
    MviWakeupReason wakeupReason;
    .....
    fd1 = open(...); fd2 = open(...);
    wdToutFd = open(...); wdFd = open(...);
    width = (fd1 > fd2) ? fd1 : fd2; width++;
    .....
    /* Initialize the read descriptor with trig condition */
    .....
    mviReqData( fd1, &readDesc );
    FD_ZERO( &readFds ); FD_SET( fd1, &readFds );
    mviSelect(width, &readFds, NULL, wdToutFd, wdFd, DxTrue);
    if (FD_ISSET(fd1,&readFds))
    {
        ioctl(fd1, MviCGetRxWakeupReason, wakeupReason);
        if (wakeupReason & MviRXCompleted)
        {
            /* mviSendData() succeeded */
            mviGetData(fd1, &readDesc)
            writeDesc.writeBuffer = readDesc.readBuffer;
            writeDesc.noOfChar = readDesc.noOfReceivedChars;
            mviSendData( fd2, &writeDesc);
            FD_ZERO( &writeFds ); FD_SET( fd2, &writeFds );
            mviSelect(width, NULL, &writeFds,
                      wdToutFd, wdFd, DxTrue);
            if (FD_ISSET(fd2,&writeFds))
            {
                ioctl(fd2, MviCGetTxWakeupReason, wakeupReason);
                if (wakeupReason & MviTXCompleted)
                {
                    /* mviSendData() succeeded */
                }
                else
                {
                    /* mviSendData() failed */
                }
            }
        }
    }
    else
    {
        /* mviReqData() failed */
    }
}
}
```

3.6.4.11 MviSendLinkStatus

NAME

mviSendLinkStatus - setting up or taking down a connection

SYNOPSIS

```
MviStatus mviSendLinkStatus(MviDL* dlPtr, MviGlobalData* globalDataPtr,  
                             MviLinkStatus newLinkStatus
```

dlPtr, a pointer to the MviDL data structure

globalDataPtr, a pointer to the MviGlobalData structure

newLinkStatus defines whether to send connect or disconnect command

The files to be included are:

- mviDLLib.h

DESCRIPTION

This routine sends a node connect or a node disconnect command to the Host Interface and waits for a response. It also updates linkStatus for the corresponding dlEntry/dlEntries accordingly.

NOTE

There is no timeout in select when waiting for the response from the Host Interface.

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.12 MviCreatePHIDevices

NAME

mviCreatePHIDevices - create Protocol Handler Interface devices

SYNOPSIS

```
MviStatus mviCreatePHIDevices(void)
```

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function creates the FIFO devices which are part of the Protocol Handler Interface.

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.13 MviCreateDLIDevices

NAME

mviCreateDLIDevices - create Data Link Handler Interface devices

SYNOPSIS

MviStatus mviCreateDLIDevices(void)

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function creates the FIFO devices which are part of the Data Link Handler Interface.

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.14 MviOpenPHIDevices

NAME

mviOpenPHIDevices - open Protocol Handler Interface devices

SYNOPSIS

MviStatus mviOpenPHIDevices(MviGlobalData* globalDataPtr, int taskInstance)

globalDataPtr, a pointer to the Protocol Handler common global data structure

taskInstance, instance number of the task spawned. Instances are numbered 1 - n.

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function opens the FIFO devices which are part of the Protocol Handler Interface.

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.15 MviOpenDLIDevices

NAME

mviOpenDLIDevices - open Data Link Handler Interface devices

SYNOPSIS

```
MviStatus mviOpenDLIDevices(int* fdDownCmndPtr,  
                             int* fdDownRespPtr,  
                             int* fdUpCmndPtr,  
                             int* fdUpRespPtr,  
                             MviHandlerType handlerType,  
                             u_int8 taskInstance)
```

fdDownCmndPtr, pointer to filedescriptor fdDownCmnd

fdDownRespPtr, pointer to filedescriptor fdDownResp

fdUpCmndPtr, pointer to filedescriptor fdUpCmnd

fdUpRespPtr, pointer to filedescriptor fdUpResp

handlerType, type of handler, DL-handler or NL-handler

taskInstance, instance number of the task spawned. Instances are numbered 1 - n.

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function opens the FIFO devices between DL-Handler and NL-Handler

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.16 MviCreateTimerDevices

NAME

mviCreateTimerDevices - create Watchdog Timer devices

SYNOPSIS

```
MviStatus mviCreateTimerDevices(u_int8 taskInstance)
```

taskInstance, instance number of the task spawned. Instances are numbered 1 - n.

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function creates the Watchdog Timer devices which are used by the Protocol Handler.
The devices are named “/wdTimer/index” where index is:

x000 - x099 PollTimers, one for each node

x100 Transmission timer

x101 Acknowledge timer

x102 System Watchdog Timer

x103 - x109 Reserved

x110 - x119 Free

x = taskInstance

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.17 MviOpenTimerDevices

NAME

mviOpenTimerDevices - open Watchdog Timer devices

SYNOPSIS

```
MviStatus mviOpenTimerDevices(MviDL* dlPtr,  
                               int* txTimerFdPtr,  
                               int* ackTimerFdPtr,  
                               int* watchdogTimerFdPtr,  
                               MviHandlerType handlerType)
```

dlPtr, a pointer to the DL data structure

txTimerFdPtr, a pointer to the txTimer file descriptor

ackTimerFdPtr, a pointer to the ackTimer file descriptor

watchdogTimerFdPtr, a pointer to the watchdogTimer file descriptor

handlerType specifies the type of handler for which the DL structure is to be initiated

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function opens the Watchdog Timer devices which are used by the Protocol Handler.
The devices opened are :

/wdTimer/x100 Transmission timer

/wdTimer/x101 Acknowledge timer

/wdTimer/x102 System Watchdog Timer

x = taskInstance.

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.18 MviFetchSetupInfo

NAME

mviFetchSetupInfo - fetch initialization information from application

SYNOPSIS

MviStatus mviFetchSetupInfo(MviDL* dlPtr, MviGlobalData* globalDataPtr)

dlPtr, a pointer to the DL data structure

globalDataPtr, a pointer to the global data structure

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function fetches initialization information, for the Protocol Handler, from the application.

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.19 MviInitDLStruct

NAME

mviInitDLStruct - initialize the DL structure

SYNOPSIS

MviStatus mviInitDLstruct(MviDL* dlPtr,
size_t noOfReadBuffers,
size_t noOfWriteBuffers,
size_t dlEntrySize,
u_int8 taskInstance,
MviHandlerType handlerType)

dlPtr, a pointer to the DL structure

noOfReadBuffers, number of read buffers used by the Protocol Handler

noOfWriteBuffers, number of write buffers used by the Protocol Handler

dlEntrySize, size of protocol specific dlEntry struct

taskInstance, instance number of the task spawned. Instances are numbered 1 - n.

handlerType specifies the type of handler for which the DL structure is to be initiated

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function allocates data structures associated with the DL structure and initializes the DL structure, common part of DL-entry, status info structure and diagnostic info structure.

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.20 MviInitGlobalDataStruct

NAME

mviInitGlobalDataStruct - initializes the common part of the GlobalData structure

SYNOPSIS

MviStatus mviInitGlobalDataStruct(MviGlobalData globalDataPtr)

globalDataPtr, a pointer to the global data structure

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function allocates data structures associated to the GlobalData structure and initializes the GlobalData structure.

RETURNS

MviOk or, if the operation fails, MviError.

3.6.4.21 MviMakeWDTDevName

NAME

mviMakeWDTDevName - creates a device name string for Watchdog Timer devices

SYNOPSIS

char* mviMakeWDTDevName(char strBuf, u_int16 devIndex)

strBuf, the buffer in which to return the string

devIndex, device instance number

The files to be included are:

- mviDLLib.h

DESCRIPTION

Returns the device name for the given parameters.

RETURNS

Pointer to the device name string.

3.6.4.22 MviPHICleanup

NAME

mviPHICleanup - frees memory for data structures allocated and closes devices opened

SYNOPSIS

```
void mviPHICleanup(MviDL* dlPtr, MviglobalData* globalDataPtr,  
                  MviHandlerType handlerType)
```

dlPtr, a pointer to the DL data structure

globalDataPtr, a pointer to the global data structure

handlerType specifies the type of handler for which the DL structure is to be initiated

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function frees memory for data structures allocated and closes devices previously opened by MVI support routines. The pointers in the MviDL struct cannot be used until a new initialization is done.

NOTE

Data structures allocated and devices opened explicitly by the Protocol Handler must be handled separately.

RETURNS

Nothing returned.

3.6.4.23 MviDLICleanup

NAME

mviDLICleanup - frees memory for data structures allocated and closes devices opened

SYNOPSIS

```
void mviDLICleanup(MviDL* dlPtr, MviglobalData* globalDataPtr,  
                  MviHandlerType handlerType)
```

dlPtr, a pointer to the DL data structure

globalDataPtr, a pointer to the global data structure

handlerType specifies the type of handler for which the DL structure is to be initiated

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function frees memory for data structures allocated and closes devices previously opened by MVI support routines. The pointers in the MviDL struct cannot be used until a new initialization is done.

NOTE

Data structures allocated and devices opened explicitly by the Protocol Handler must be handled separately.

RETURNS

Nothing returned.

3.6.4.24 MviPHIWaitForModemSignals

NAME

mviPHIWaitForModemSignals - waits for modem signals to become active

SYNOPSIS

DxBoolean mviPHIWaitForModemSignals(MviDL* dlPtr, MviGlobalData* globalDPtr)

dlPtr, a pointer to the DL data structure

globalDPtr, a pointer to the global data structure

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function waits for the modem signals DCD CTS and DSR to become active in a Protocol Handler which is not divided into separate Network and Data Link Layers. Depending on if the communication is made over a dial-up line or the UART is working in half or full duplex mode, different combinations of the modem signals are waited for:

- Dial-up line: either one of DSR or CTS signal are waited for.
- Half Duplex: DSR is waited for.
- Full Duplex: all of DCD CTS and DSR are waited for.

This function also supervise PHI Down Stream and Local commands. All Down Stream commands are responded to with response code "MviCRspHandlerBusy". All Local commands except "MviRestartHnd" are responded to with response code "MviCRspHandlerBusy". If a Local command "MviRestartHnd" is received, a response is sent with response code "MviCRspOk" and the function terminates.

RETURNS

When the proper modem signals are active, the function returns DxTrue. If a Local command "MviRestartHnd" is received, the function returns DxFalse.

3.6.4.25 MviDLIWaitForModemSignals

NAME

mviDLIWaitForModemSignals - waits for modem signals to become active

SYNOPSIS

DxBoolean mviDLIWaitForModemSignals(MviDL* dlPtr, MviGlobalData* globalDPtr)

dlPtr, a pointer to the DL data structure

globalDPtr, a pointer to the global data structure

The files to be included are:

- mviDLLib.h

DESCRIPTION

This function waits for the modem signals DCD CTS and DSR to become active, in the Data Link Layer of a Protocol Handler which is divided into separate Network and Data Link Layers. Depending on if the communication is made over a dial-up line or the UART is working in half or full duplex mode, different combinations of the modem signals are waited for:

- Dial-up line: either one of DSR or CTS signal are waited for.
- Half Duplex: DSR is waited for.
- Full Duplex: all of DCD CTS and DSR are waited for.

This function also supervise DLI Down commands. All DLI Down commands except "MviDLISetupInfo" are responded to with response code "MviDLIRspDLNotReady". If a DLI Down command "MviDLISetupInfo" is received, a response is sent with response code "MviDLIRspOk" and the function terminates.

RETURNS

When the proper modem signals are active, the function returns DxTrue. If a DLI Down command "MviDLISetupInfo" is received, the function returns DxFalse.

3.6.4.26 MviSysMess

NAME

mviSysMess - send a system message

SYNOPSIS

```
void mviSysMess(int32 component, int32 msgNum,  
                int32 data1, int32 data2, char* text,)
```

component identifies the software unit which discovered the error. This should be one of the MVI components defined in vxbsSysMess.h.

msgNum, error code

data1, field 1 for additional information

data2, field 2 for additional information

text, a text string which can be used for a description of the error

The files to be included are:

- mviSysMess.h
- vxbsSysMess.h

DESCRIPTION

This routine sends a system message. All the data is copied into the system message queue, so there is no need to preserve the arguments after the call of mviSysMess.

If the system message queue is full, the message is lost and an error message is printed on stderr. When there is room in the message queue again, a system message indicating that one or more messages have been lost is sent.

If the message queue is filled to the 80 percent mark or higher, a system message saying that the queue is almost full is sent.

In Advant Controller 400, the system message is sent to the process station, mapped on a CX system message and sent to the system message queue. Component is mapped on Mcode, msgNum is mapped on trace number, data1 and data2 are mapped on data1 and data2, respectively, and text is discarded.

RETURNS

Nothing returned.

3.6.5 MVI Protocol Handler Interface

MVI Protocol Handler Interface provides a set of functions that encapsulate the inter-task communication between the Protocol Handler and the Host Interface, see [Figure 3-14](#).

The following functionality is provided to comprise the interface between the Protocol Handler and Host Interface:

- Send Diagnostics Information
 - This function sends Diagnostic Information to the Host Interface. This information includes communication link quality such as number of modem signal errors, parity errors, checksum errors, framing errors, character timeout errors and number of re-transmissions.
- Send Watchdog Messages
 - This function sends Watchdog messages to the Host Interface. These messages are sent to the Host and can be used to monitor the Protocol Handler, whether it is alive or not.
- Send Status Information
 - This function sends Status Information to the Host Interface. The Status Information includes Link Status, which tells whether the PLC:s on the communication link are reachable. Most of the Status Information is protocol specific.
- Read setup parameters / Send setup parameters
 - These functions are used by the Protocol Handler to make a request for, and receive setup information from the Host System. This information includes the line characteristics used to initialize the UART. It also includes information about the PLC:s on the communication link.
- Local Command / Local Response
 - These functions are used by the Protocol Handler to receive and respond to commands directed towards the Protocol Handler itself. The type of commands sent are control commands, see [Table 3-2](#).
- Down Stream Command / Down Stream Response
 - These functions are used by the Protocol Handler to receive and respond to read and write commands directed towards a remote node on the communication link. See [Table 3-2](#).
- Up Stream Command / Up Stream Response.
 - These functions are used by the Protocol Handler to send and wait for response of read and write commands directed towards the Host System. These commands are built by the Protocol Handler when receiving a command message from a remote node on the communication link. See [Table 3-2](#).

The inter-task communication is handled using pipes ¹.

1. A pipe is a data structure used for inter-task communication.

The functions that provide the functionality described above, see [Section 3-14, MVI Protocol Handler Interface](#), result in a data/command message exchange between the Protocol Handler and the Host Interface. Each of the functions described above will have a dedicated pipe, to allow a flexible flow control between the Host Interface and the Protocol Handler.

The Host Interface handles the communication with the Host System. It also handles data and information conversion between the Host System and the Protocol Handler Interface.

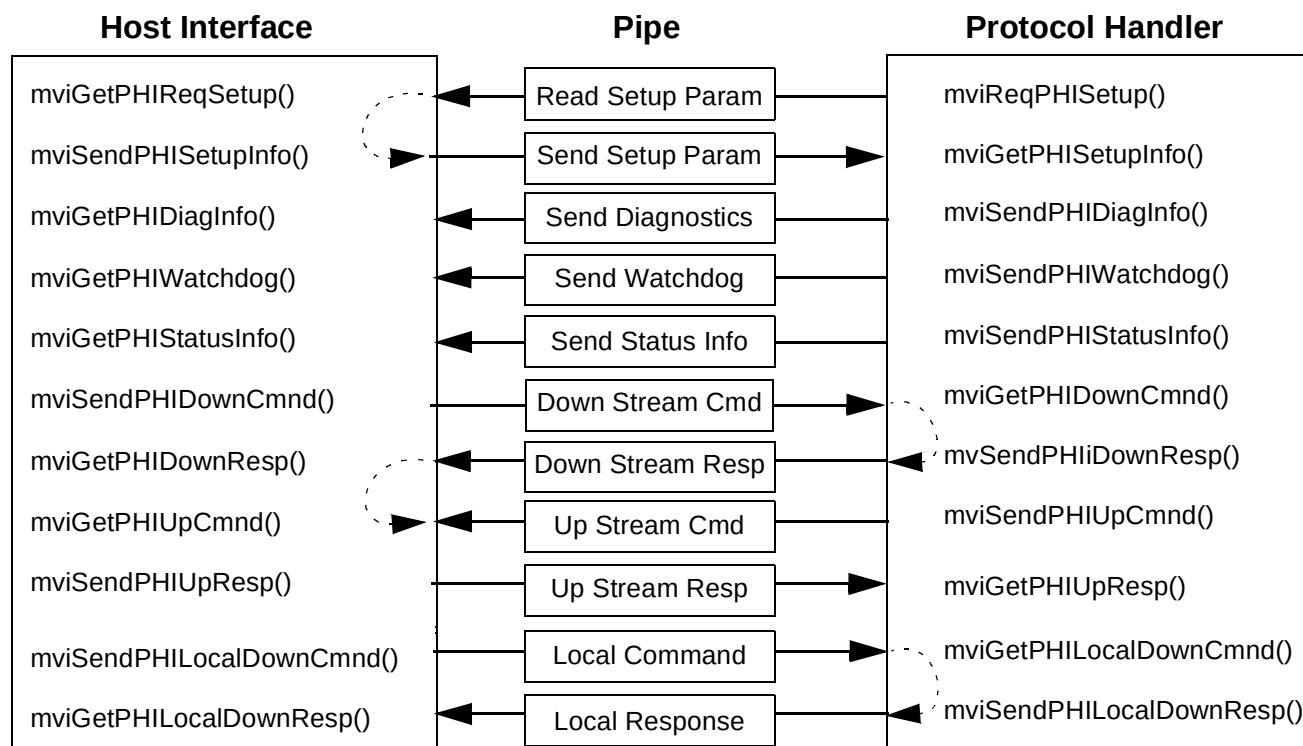


Figure 3-14. Interface between Protocol Handler and Host Interface

There are two types of Protocol Handler Interface functions: information functions used by the Protocol Handler to send information to the Host Interface and functions that rely on a command/response protocol that must be fulfilled.

The following functions do not use a command / response protocol:

- Send Diagnostics
 - The Protocol Handler sends a Diagnostics Information message to the Host Interface using the function **mviSendPHIDiagInfo**.
- Send Watchdog
 - The Protocol Handler sends a Watchdog message to the Host Interface using the function **mviSendPHIWatchdog**.

- Send Status Info
 - The Protocol Handler sends a Status Information message to the Host Interface using the function **mviSendPHIStatusInfo**.

The following functions rely on a command / response protocol, which must be fulfilled:

- Read Setup Param / Send Setup Param
 - The Protocol Handler sends a Read Setup Param request to the Host Interface using the function **mviPHIReqSetup** and waits for the response. After receiving the request, the Host Interface sends the setup parameters. The Protocol Handler receives the setup parameters which are fetched using the function **mviGetPHISetupInfo**.
- Down Stream Command / Down Stream Response
 - The Protocol Handler receives a Down Stream Command from the Host Interface, which is fetched using the function **mviGetPHIDownCmnd**. After the command has been handled, the Protocol Handler sends a response to the Host Interface using the function **mviSendPHIDownResp**.
- Up Stream Command / Up Stream Response
 - The Protocol Handler sends an Up Stream Command to the Host Interface using the function **mviSendPHIUpCmnd** and waits for a response. After the command has been handled, the Protocol Handler receives a response message, which is fetched using the function **mviGetPHIUpResp**.
- Local Command / Local Response
 - The Protocol Handler receives a Local Command from the Host Interface, which is fetched using the function **mviGetPHILocalDownCmnd**. After the command has been handled, the Protocol Handler sends a response to the Host Interface using the function **mviSendPHILocalDownResp**.

Table 3-2 below describes the Commands supported by MVI Protocol Handler Interface.

Table 3-2. MVI Protocol Handler Interface Commands

Command Type	Command Code	Name	Description
Control Commands (Local Commands)	1	MviRestartHandler	Re-starts the Protocol Handler.
	2	MviResetDiag	Resets the Diagnostics Information Structure.
	3	MviDialStation	Tells the Protocol Handler to dial a remote node
	4	MviHangupLine	Tells the Protocol Handler to hang up the line to a remote node
	5	MviNodeConn	Used by the Protocol Handler to inform the Host that the connection with a remote node is established.
	6	MviNodeDisConn	Used by the Protocol Handler to inform the Host that the connection with a remote node is lost.
	7	MviGetDiag	Fetches the Diagnostics Information Structure.

Table 3-2. MVI Protocol Handler Interface Commands (Continued)

Command Type	Command Code	Name	Description
Read Commands (Downstream and Upstream Commands)	101	MviRdSnglInt16	Read one 16 bit integer value from a remote node in case of a Downstream Command, or from the Host System in case of an Upstream Command.
	102	MviRdMultInt16	Read a number of 16 bit integer values from a remote node in case of a Downstream Command, or from the Host System in case of an Upstream Command.
	103	MviRdSnglInt32	Read one 32 bit integer value from a remote node in case of a Downstream Command, or from the Host System in case of an Upstream Command.
	104	MviRdMultInt32	Read a number of 32 bit integer values from a remote node in case of a Downstream Command, or from the Host System in case of an Upstream Command.
	105	MviRdSnglBool	Read one boolean value from a remote node in case of a Downstream Command, or from the Host System in case of an Upstream Command. See Figure 3-15 for the byte order of boolean data in the Application Data field of the MviCmndInfo structure.
	106	MviRdMultBool	Read a number of boolean values from a remote node in case of a Downstream Command, or from the Host System in case of an Upstream Command. See Figure 3-15 for the byte order of boolean data in the Application Data field of the MviCmndInfo structure.
	107	MviRdSnglFloat	Read one float value from a remote node in case of a Downstream Command, or from the Host System in case of an Upstream Command.
	108	MviRdMultFloat	Read a number of float values from a remote node in case of a Downstream Command, or from the Host System in case of an Upstream Command.
	109	MviRdString	Read one string value from a remote node in case of a Downstream Command, or from the Host System in case of an Upstream Command.
	110	MviRdMix32	Read a number of 32 bit values from a remote node in case of a Downstream Command, or from the Host System in case of an Upstream Command.

Table 3-2. MVI Protocol Handler Interface Commands (Continued)

Command Type	Command Code	Name	Description
Write Commands (Downstream and Upstream Commands)	201	MviWrSnglInt16	Write one 16 bit integer value to a remote node in case of a Downstream Command, or to the Host System in case of an Upstream Command.
	202	MviWrMultInt16	Write a number of 16 bit integer values to a remote node in case of a Downstream Command, or to the Host System in case of an Upstream Command.
	203	MviWrSnglInt32	Write one 32 bit integer value to a remote node in case of a Downstream Command, or to the Host System in case of an Upstream Command.
	204	MviWrMultInt32	Write a number of 32 bit integer values to a remote node in case of a Downstream Command, or to the Host System in case of an Upstream Command.
	205	MviWrSnglBool	Write one boolean value to a remote node in case of a Downstream Command, or to the Host System in case of an Upstream Command. See Figure 3-15 for the byte order of boolean data in the Application Data field of the MviCmndInfo structure.
	206	MviWrMultBool	Write a number of boolean values to a remote node in case of a Downstream Command, or to the Host System in case of an Upstream Command. See Figure 3-15 for the byte order of boolean data in the Application Data field of the MviCmndInfo structure.
	207	MviWrSnglFloat	Write one float value to a remote node in case of a Downstream Command, or to the Host System in case of an Upstream Command.
	208	MviWrMultFloat	Write a number of float values to a remote node in case of a Downstream Command, or to the Host System in case of an Upstream Command.
	209	MviWrString	Write one string value to a remote node in case of a Downstream Command, or to the Host System in case of an Upstream Command.
	210	MviWrMix32	Write a number of 32 bit values to a remote node in case of a Downstream Command, or to the Host System in case of an Upstream Command.

NOTE

The MVI Protocol Handler Interface commands MviRdString and MviWrString are currently not supported.

The byte order for multiple boolean data in the Application Data field of the MviCmndInfo Data Structure is shown in Figure 3-15. For a single boolean value bit 0 of byte 0 is used. That is, for a boolean value of 0, bit 0 of byte 0 is cleared. For a boolean value of 1, bit 0 of byte 0 is set.

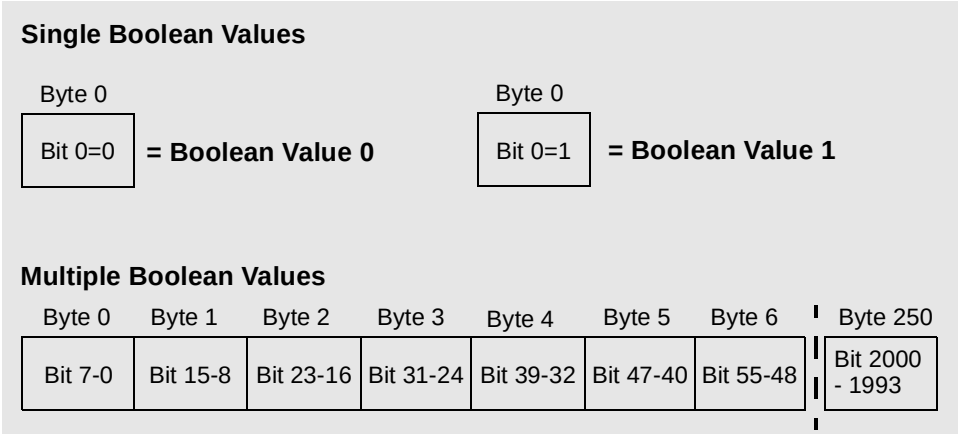


Figure 3-15. Application Data field of the MviCmndInfo Data Structure

For a description of data structures, refer to the include file mviProtInt.h which is part of MVIBase.

3.6.5.1 MviReqPHISetup

NAME

mviReqPHISetup - sends a request message from Protocol Handler to the Host Interface

SYNOPSIS

MviStatus mviReqPHISetup(int fileDescr, MviSetupReqMess **suMessage)
fileDescr, file descriptor for the pipe
suMessage, pointer to pointer to a setup request message struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function sends a request message for setup parameters from the Protocol Handler to the Host Interface. If mviReqSetup is called, the same function may not be called again until a response is received by mviGetSetupInfo.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.5.2 MviGetPHIReqSetup

NAME

mviGetPHIReqSetup - receives a request message for setup parameters from the Protocol Handler

SYNOPSIS

```
MviStatus mviGetPHIReqSetup(int fileDescr, MviSetupReqMess **suMessage)
```

fileDescr, file descriptor for the pipe

suMessage, pointer to pointer to a setup request message struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function receives a request message for setup parameters from the Protocol Handler.

RETURNS

MviOk or, if the operation fails, MviReadError.

3.6.5.3 MviSendPHISetupInfo

NAME

mviSendPHISetupInfo - sends a setup response message from Host Interface to Protocol Handler

SYNOPSIS

```
MviStatus mviSendPHISetupInfo(int fileDescr, MviProtSetupInfo **setupData)
```

fileDescr, file descriptor for the pipe

setupData, a pointer to a pointer to parameter struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function sends a setup parameter response message from the Host Interface to the Protocol Handler.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.5.4 MviGetPHISetupInfo

NAME

mviGetPHISetupInfo - receives setup parameters from the Host Interface

SYNOPSIS

MviStatus mviGetPHISetupInfo(int fileDescr, MviProtSetupInfo **setupData)

fileDescr, filedescriptor for the pipe

setupData, a pointer to a pointer to parameter struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Protocol Handler to receive setup parameters from the Host Interface.

RETURNS

MviOk or, if the operation fails, MviReadError.

3.6.5.5 MviSendPHIWatchdog

NAME

mviSendPHIWatchdog - sends a Watchdog message from Protocol Handler to Host Interface

SYNOPSIS

MviStatus mviSendPHIWatchdog(int fileDescr, MviWatchdogMess **wdMessage)

fileDescr, file descriptor for the pipe

wdMessage, pointer to a pointer to a Watchdog message

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Protocol Handler to send a Watchdog message to the Host Interface.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.5.6 MviGetPHIWatchdog

NAME

mviGetPHIWatchdog - receives a Watchdog message from the Protocol Handler

SYNOPSIS

MviStatus mviGetPHIWatchdog(int fileDescr, MviWatchdogMess **wdMessage)

fileDescr, file descriptor for the pipe

wdMessage, pointer to a pointer to a Watchdog message

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Host Interface to receive a Watchdog message from the Protocol Handler.

RETURNS

MviOk or, if the operation fails, MviReadError.

3.6.5.7 MviSendPHIDiagInfo

NAME

mviSendPHIDiagInfo - sends a diagnostic information to the Host Interface

SYNOPSIS

MviStatus mviSendPHIDiagInfo(int fileDescr, MviDiagInfo **diagData)

fileDescr, file descriptor for the pipe

diagData, a pointer to a pointer to the data struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Protocol Handler to send the diagnostic information to the Host Interface.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.5.8 MviGetPHIDiagInfo

NAME

mviGetPHIDiagInfo - receives diagnostic information from the Protocol Handler

SYNOPSIS

MviStatus mviGetPHIDiagInfo(int fileDescr, MviDiagInfo **diagData)

fileDescr, file descriptor for the pipe

diagData, a pointer to a pointer to the data struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Host Interface to receive diagnostic information from the Protocol Handler.

RETURNS

MviOk or, if the operation fails, MviReadError.

3.6.5.9 MviSendPHIStatusInfo

NAME

mviSendPHIStatusInfo - sends status message to the Host Interface

SYNOPSIS

MviStatus mviSendPHIStatusInfo(int fileDescr, MviNodeStat **statusInfo)

fileDescr, file descriptor for the pipe

statusInfo, a pointer to a pointer to the info struct for the node in question

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Protocol Handler to send a status change message to the Host Interface. Status change messages contain information about the communication link status (up/down), illegal requests and illegal use of functions.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.5.10 MviGetPHIStatusInfo

NAME

mviGetPHIStatusInfo - receive status information from the Protocol Handler

SYNOPSIS

```
MviStatus mviGetPHIStatusInfo(int fileDescr, MviNodeStat **statusInfo)
```

fileDescr, file descriptor for the pipe

statusInfo, a pointer to a pointer to the info struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Host Interface to receive status information from the Protocol Handler.

RETURNS

MviOk or, if the operation fails, MviReadError.

3.6.5.11 MviSendPHILocalDownCmnd

NAME

mviSendPHILocalDownCmnd - send a local command message to the Protocol Handler

SYNOPSIS

```
MviStatus mviSendPHILocalDownCmnd(int fileDescr, MviCmndInfo **command)
```

fileDescr, file descriptor for the pipe

command, a pointer to a pointer to the command info struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Host Interface to send a local command message to the Protocol Handler. If mviSendPHILocalDownCmnd is called, the same function must not be called again until a response has been received by mviGetPHILocalDownResp.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.5.12 MviGetPHILocalDownCmnd

NAME

mviGetPHILocalDownCmnd - receives a local command message from the Host Interface

SYNOPSIS

```
MviStatus mviGetPHILocalDownCmnd(int fileDescr, MviCmndInfo **command)
fileDescr, file descriptor for the pipe
command, a pointer to a pointer to the command info struct
```

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Protocol Handler to receive a local command message from the Host Interface.

RETURNS

MviOk or, if the operation fails, MviReadError.

3.6.5.13 MviSendPHILocalDownResp

NAME

mviSendPHILocalDownResp - sends a local response messages to the Host Interface

SYNOPSIS

```
MviStatus mviSendPHILocalDownResp(int fileDescr, MviCmndInfo **response)
fileDescr, file descriptor for the pipe
response, a pointer to a pointer to the respond info struct
```

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Protocol Handler to send a local response message to the Host Interface.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.5.14 MviGetPHILocalDownResp

NAME

mviGetPHILocalDownResp - receives a local response message from the Protocol Handler

SYNOPSIS

MviStatus mviGetPHILocalDownResp(int fileDescr, MviCmndInfo **response)

fileDescr, file descriptor for the pipe

response, a pointer to a pointer to the response info struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Host Interface to receive a local response message from the Protocol Handler.

RETURNS

MviOk or, if the operation fails, MviReadError.

3.6.5.15 MviSendPHIDownCmnd

NAME

mviSendPHIDownCmnd - sends a Down Stream command message to the Protocol Handler

SYNOPSIS

MviStatus mviSendPHIDownCmnd(int fileDescr, MviCmndInfo **command)

fileDescr, file descriptor for the pipe

command, a pointer to a pointer to the command info struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Host Interface to send a Down Stream command messages to the Protocol Handler. If MviSendPHIDownCmnd is called, the same function must not be called again until a response is received by mviGetPHIDownResp.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.5.16 MviGetPHIDownCmnd

NAME

mviGetPHIDownCmnd - receives a Down Stream command message from the Host Interface

SYNOPSIS

MviStatus mviGetPHIDownCmnd(int fileDescr, MviCmndInfo **command)

fileDescr, file descriptor for the pipe

command, a pointer to a pointer to the command info struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Protocol Handler to receive a Down Stream command messages from the Host Interface.

RETURNS

MviOk or, if the operation fails, MviReadError.

3.6.5.17 MviSendPHIDownResp

NAME

mviSendPHIDownResp - send a Down Stream response message to the Host Interface

SYNOPSIS

MviStatus mviSendPHIDownResp(int fileDescr, MviCmndInfo **response)

fileDescr, file descriptor for the pipe

response, a pointer to a pointer to the response info struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Protocol Handler to send a Down Stream response message to the Host Interface.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.5.18 MviGetPHIDownResp

NAME

mviGetPHIDownResp - receives a Down Stream response message from the Protocol Handler

SYNOPSIS

MviStatus mviGetPHIDownResp(int fileDescr, MviCmndInfo **response)

fileDescr, file descriptor for the pipe

response, a pointer to a pointer to the response info struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Host Interface to receive a Down Stream response message from the Protocol Handler.

RETURNS

MviOk or, if the operation fails, MviReadError.

3.6.5.19 MviSendPHIUpCmnd

NAME

mviSendPHIUpCmnd - sends an Upstream Command message to the Host Interface

SYNOPSIS

MviStatus mviSendPHIUpCmnd(int fileDescr, MviCmndInfo **command)

fileDescr, file descriptor for the pipe

command, a pointer to a pointer to the command info struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Protocol Handler to send an Upstream Command message to the Host Interface. If MviSendPHIUpCmnd is called, the same function must not be called again until a response is received by MviGetPHIUpResp.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.5.20 MviGetPHIUpCmnd

NAME

mviGetPHIUpCmnd - receives an Upstream Command message from the Protocol Handler

SYNOPSIS

MviStatus mviGetPHIUpCmnd(int fileDescr, MviCmndInfo **command)

fileDescr, file descriptor for the pipe

command, a pointer to a pointer to the command info struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Host Interface to receive an Upstream Command message from the Protocol Handler.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.5.21 MviSendPHIUpResp

NAME

mviSendPHIUpResp - sends an Upstream Response message to the Protocol Handler

SYNOPSIS

MviStatus mviSendPHIUpResp(int fileDescr, MviCmndInfo **response)

fileDescr, file descriptor for the pipe

response, a pointer to a pointer to the response info struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Host Interface to send an Upstream Response message to the Protocol Handler.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.5.22 MviGetPHIUpResp

NAME

mviGetPHIUpResp - receives an Upstream Response message from the Host Interface

SYNOPSIS

```
MviStatus mviGetPHIUpResp(int fileDescr, MviCmndInfo **response)
```

fileDescr, file descriptor for the pipe

response, a pointer to a pointer to the response info struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Protocol Handler to receive an Upstream Response message from the Host Interface.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.6 MVI DataLink Interface

MVI Data Link Interface provides a set of functions that encapsulates the inter-task communication between the Network Handler and the Data Link Handler, in case of a Protocol Handler split into two tasks, see [Figure 3-16](#).

The following functionality is provided to comprise the interface between the Network Handler task and the Data Link Handler task in a split Protocol Handler:

- DLI Down Command / DLI Down Response
 - These functions are used to send commands from the Network Handler to the Data Link Handler and responses in the opposite direction.
- DLI Up Command / DLI Up Response.
 - These functions are used to send commands from the Data Link Handler to the Network Handler and responses in the opposite direction.

The inter-task communication is handled using pipes ¹.

The functions that provide the functionality described above, see [Figure 3-16](#), result in a data/command message exchange between the Network Handler and the Data Link Handler. Each of the functions listed above will have a dedicated pipe to allow a flexible flow control between the two tasks.

1. A pipe is a data structure used for inter task communication.

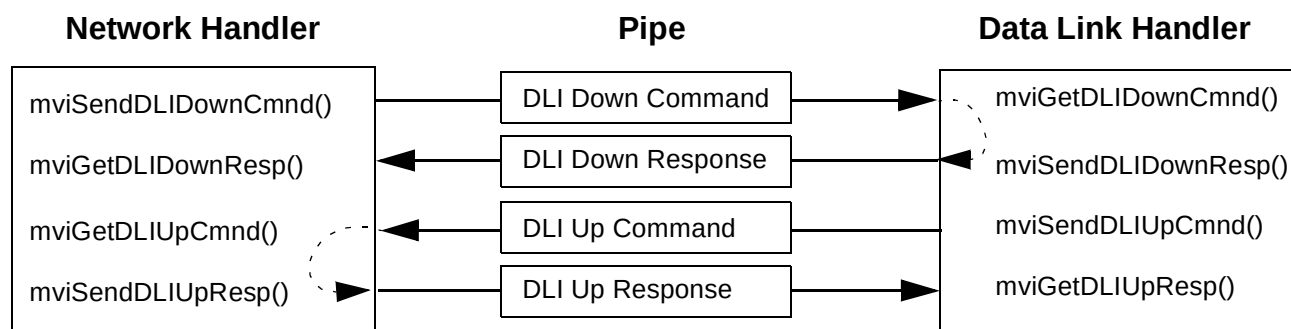


Figure 3-16. Interface between Network Handler and Data Link Handler

The Data Link Interface functions rely on a command / response protocol that must be fulfilled:

- DLI Down Command / DLI Down Response
 - The Network Handler sends a command to the Data Link Handler using the function **mviSendDLIDownCmnd**, and waits for a response. The Data Link Handler fetches the command using the function **mviGetDLIDownCmnd**. After the command has been handled, the Data Link Handler sends a response message back to the Network Handler using the function **mviSendDLIDownResp**. The Network Handler fetches the response message using the function **mviGetDLIDownResp**.
- DLI Up Command / DLI Up Response.
 - The Data Link Handler sends a command to the network Handler using the function **mviSendDLIUpCmnd**, and waits for a response. The Network Handler fetches the command using the function **mviGetDLIUpCmnd**. After the command has been handled, the Network Handler sends a response message back to the Data Link Handler using the function **mviSendDLIUpResp**. The Data Link Handler fetches the response message using the function **mviGetDLIUpResp**.

Table 3-3 below describes the commands supported by MVI Data Link Interface.

Table 3-3. MVI Data Link Interface Commands

Command Code	Name	Description
1	MviDLIRestartDL	DLI Down Command: The Network handler uses this command code to Re-start the Data Link Handler.
2	MviDLISetupInfo	DLI Down Command: The Network handler uses this command code to send down the setup information for the UART, to the Data Link Handler
3	MviDLIDataBuffer	DLI Down Command: The Network Handler sends a data message, down to the Data Link Handler, addressed to a remote node on the communication link. DLI Up Command: The Data Link Handler sends a data message, coming from a remote node on the communication link, up to the Network Handler.
4	MviDLIStatusReport	DLI Up Command: The Data Link Handler sends a status report message up to the Network Handler
5	MviDLILocalCmnd	DLI Down Command: The Network Handler sends a local command down to the Data Link handler. This command code uses a sub-code, local command code, which is protocol specific, that is, with this command code the implementor can define any functionality.

For a description of data structures, refer to the include file mviProtInt.h which is part of MVIBase.

3.6.6.1 MviSendDLIDownCmnd

NAME

mviSendDLIDownCmnd - sends a DLI Down Stream command message to the Data Link Handler

SYNOPSIS

```
MviStatus mviSendDLIDownCmnd(int fileDescr, MviDLICmndInfo **command)
```

fileDescr, file descriptor for the pipe

command, a pointer to a pointer to the DLI command info struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Network Handler to send a DLI Down Stream command message to the Data Link Handler. If this function is called, it should not be called again until a response has been received by mviGetDLIDownResp.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.6.2 MviGetDLIDownCmnd

NAME

mviGetDLIDownCmnd - receives a DLI Down Stream command message from the Network Handler

SYNOPSIS

```
MviStatus mviGetDLIDownCmnd(int fileDescr, MviDLICmndInfo **command)
fileDescr, file descriptor for the pipe
command, a pointer to a pointer to the DLI command info struct
```

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Data Link Handler to receive a DLI Down Stream command message from the Network Handler.

RETURNS

MviOk or, if the operation fails, MviReadError.

3.6.6.3 MviSendDLIDownResp

NAME

mviSendDLIDownResp - sends a DLI Down Stream response message to the Network Handler

SYNOPSIS

```
MviStatus mviSendDLIDownResp(int fileDescr, MviDLICmndInfo **response)
fileDescr, file descriptor for the pipe
response, a pointer to a pointer to the command info struct
```

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Data Link Handler to send a DLI Down Stream response message to the Network Handler.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.6.4 MviGetDLIDownResp

NAME

mviGetDLIDownResp - receives a DLI Down Stream response message from the Data Link Handler

SYNOPSIS

MviStatus mviGetDLIDownResp(int fileDescr, MviDLCmdInfo **response)

fileDescr, file descriptor for the pipe

response, a pointer to a pointer to the coammand info struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Network Handler to receive a DLI Down Stream response message from the Data Link Handler.

RETURNS

MviOk or, if the operation fails, MviReadError.

3.6.6.5 MviSendDLIUpCmd

NAME

mviSendDLIUpCmd - sends a DLI Up Stream command message to the Network Handler

SYNOPSIS

MviStatus mviSendDLIUpCmd(int fileDescr, MviDLICmdInfo **command)

fileDescr, file descriptor for the pipe

command, a pointer to a pointer to the DLI command info struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Data Link Handler to send DLI Up Stream command messages to the Network Handler. If this function is called, it should not be called again until a response has been received by mviGetDLIUpResp.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.6.6 MviGetDLIUpCmnd

NAME

mviGetDLIUpCmnd - receives a DLI Up Stream command message from the Data Link Handler

SYNOPSIS

```
MviStatus mviGetDLIUpCmnd(int fileDescr, MviDLICmndInfo **command)
fileDescr, file descriptor for the pipe
command, a pointer to a pointer to the DLI command info struct
```

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Network Handler to receive a DLI Up Stream command message from the Data Link Handler.

RETURNS

MviOk or, if the operation fails, MviReadError.

3.6.6.7 MviSendDLIUpResp

NAME

mviSendDLIUpResp - sends a DLI Up Stream response message to the Data Link Handler

SYNOPSIS

```
MviStatus mviSendDLIUpResp(int fileDescr, MviDLICmndInfo **response)
fileDescr, file descriptor for the pipe
response, a pointer to a pointer to the command info struct
```

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Network Handler to send a DLI response messages to the Data Link Handler.

RETURNS

MviOk or, if the operation fails, MviWriteError.

3.6.6.8 MviGetDLIUpResp

NAME

mviGetDLIUpResp - receives a DLI Up Stream response message from the from the network Handler

SYNOPSIS

MviStatus mviGetDLIUpResp(int fileDescr, MviDLCmndInfo **response)

fileDescr, file descriptor for the pipe

response, a pointer to a pointer to the coammand info struct

The files to be included are:

- mviProtInt.h

DESCRIPTION

This function is used by the Data Link Handler to receive a DLI Up Stream response message from the Network Handler.

RETURNS

MviOk or, if the operation fails, MviReadError.

3.6.7 Protocol Configuration and Application Building for Advant Controller 400

3.6.7.1 MS for MVI Protocol Configuration

This section describes the Configuration MS used to configure an MVI protocol. These MS contain setup information for the CI535's hardware and software. Status and Diagnostics information is also received from an MVI protocol. The Status and Diagnostics information can be used by the application program.

Line Characteristics MS

Line characteristics are specified in one MS for each asynchronous communication port, which the MVI protocol reads during its start-up/restart and start-up of Advant Controller 400. The application program can also initiate a port by sending Line Characteristics MS with the SENDREQ PC element.

NOTE

Line Characteristics MS are always blocked at runtime. If not, the MVI protocol communication software restarts each time they are received by the submodule. The submodule must be restarted (see [Section 2.4, Start-up Procedures](#)) if references in the Network Configuration MS are changed (see description later in this section). Sending the Line Characteristics MS to the MVI protocol for new initiation of the port is **not** enough.

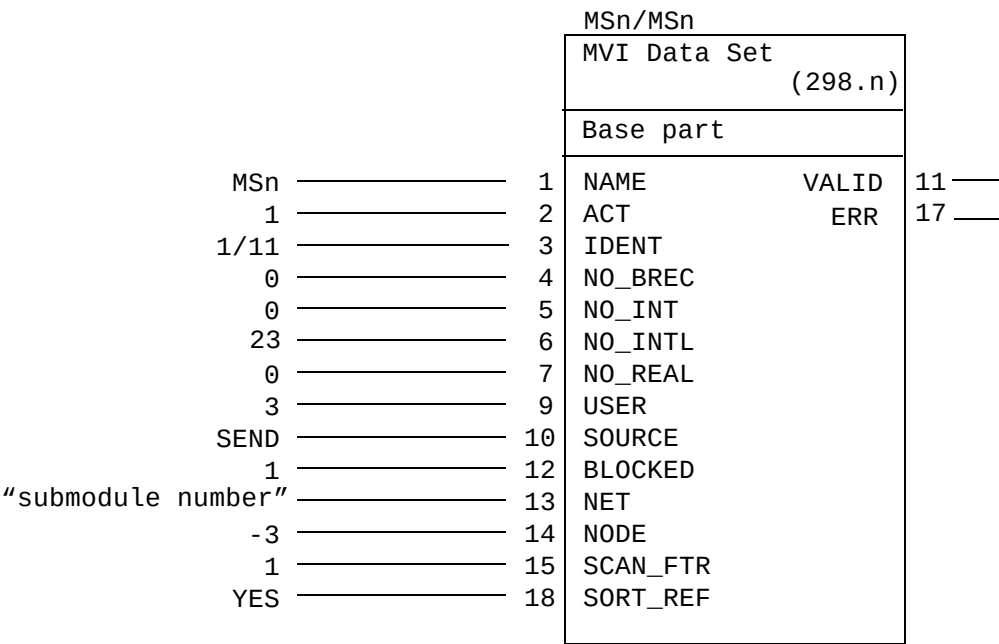


Figure 3-17. MVI Data Set for Line Characteristics Base Part

			Value references
type of RTU	_____	8(1)	REF1
master/slave	_____	8(2)	REF2
bitrate	_____	8(3)	REF3
char. length	_____	8(4)	REF4
stopbits	_____	8(5)	REF5
parity	_____	8(6)	REF6
duplex	_____	8(7)	REF7
dial	_____	8(8)	REF8
pre-idle time	_____	8(9)	REF9
carrier delay	_____	8(10)	REF10
char. time-out	_____	8(11)	REF11
turn-around time	_____	8(12)	REF12
UART time-out	_____	8(13)	REF13
retransmissions	_____	8(14)	REF14
poll cycle time	_____	8(15)	REF15
parameter 1	_____	8(16)	REF16
parameter 2	_____	8(17)	REF17
parameter 3	_____	8(18)	REF18
parameter 4	_____	8(19)	REF19
parameter 5	_____	8(20)	REF20
parameter 6	_____	8(21)	REF21
parameter 7	_____	8(22)	REF22
parameter 8	_____	8(23)	REF23

Figure 3-18. MVI Data Set for Line Characteristics Reference Part

The parameters REF1 - REF23 are stored in the DAT elements referenced by the Line Characteristics MS. You can give DAT elements arbitrary names.

The parameters have the following significance:

Table 3-4. Description of MS for Line Characteristics

Terminal	Parameter Name	Recommended Value (Min-Max)	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT		1 or 11	Identity of the MVI Data Set 1 for port 1 11 for port 2
NO_BREC		0	
NO_INT		0	
NO_INTL		23	Number of Integer Long DAT references
NO_REAL		0	
USER		3	
SOURCE		SEND	
BLOCKED		1	Note: The MS must be blocked. MS requested by the CI535 module at start-up.
NET		7 - 11	CI535 submodule number. Record number for the CI535 data base element for the MVI protocol port defines the slave module number. Record No. 1 => submodule = 7 Record No. 2 => submodule = 8 Record No. 3 => submodule = 9 (only valid for AC 450) Record No. 4 => submodule = 10 (only valid for AC 450) Record No. 5 => submodule = 11 (only valid for AC 450)
NODE		-3	Always -3 for Configuration MS
SCAN_FTR		1	Without function when BLOCKED=1
SORT_REF		YES	REF1 - 24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID			1 after first successful transmission. Reset by user.
ERR			1 if the transmission has failed due to lost contact with destination node or queue full to CI535
REF1	Type of RTU	20	Protocol type

Table 3-4. Description of MS for Line Characteristics (Continued)

Terminal	Parameter Name	Recommended Value (Min-Max)	Description
REF2	Slave/Master	0 or 1	0 = The port is a Slave 1 = The port is a Master
REF3	Bitrate		Transmission speed: 150, 300, 600, 1200, 2400, 4800, 9600 or 19200 bits/s.
REF4	Character length	7 or 8	The number of bits/character
REF5	Stopbits	10	5 = 0.5 Stop bits 10 = 1 Stop bit 15 = 1.5 Stop bits 20 = 2 Stop bits
REF6	Parity	2	0 = No parity 1 = Odd parity 2 = Even parity
REF7	Duplex	1	0 = Half duplex 1 = Full duplex 2 = Half duplex and ignore DCD
REF 8	Dial	0 or 1	Dialing. 0 = Disabled 1 = Enabled
REF9	Pre-idle time	1 half duplex 0 full duplex (0 - 255)	Pre-idle time in number of character transmission times. Time to allow the carrier wave to stabilize before transmission of the first character. Restrictions in half duplex: “(own) pre-idle. time >= (opposite side) post-idle time” See also Figure 3-19 below.
REF10	Post-idle time	1 half duplex 0 full duplex (0 - 255)	Post-idle time in number of character transmission times. Time to wait after transmission of the last character before deactivating RTS. This delay is used to avoid destruction of the last character in the message, due to lost carrier. See also Figure 3-19 below.
REF11	Char Time-out	3 (0 - 255)	The number of character transmission times to wait until the message is considered interrupted.

Table 3-4. Description of MS for Line Characteristics (Continued)

Terminal	Parameter Name	Recommended Value (Min-Max)	Description
REF12	Turn-around time	100 (5 - 10000)	Time in milliseconds to wait from the last character in the command until the first character in the reply, i.e., time-out time, where the master waits for a response from the slave node. Note: This time is dependent on the times defined in REF8 and REF9 (or corresponding set-up) in the slave node and must be adjusted accordingly. You must also include delays that may occur in the slave units. See also Figure 3-20 below.
REF 13	UART timeout	2 for half duplex 1 for full duplex (0 - 255)	UART transmission time-out in number of character transmission times. If half duplex is used, a check of DCD will be made by the send routine before activating RTS. If DCD is activated the send routine will wait for DCD to be deactivated up to this time before aborting the transmission..
REF14	Retransmission s	2 for Master 0 for Slave (0 - 200)	Max. No. of retransmissions before the line is considered broken.
REF15	Poll cycle time	96000/Bitrate for Master 1.2*96000/ Bitrate for Slave (5 - 770)	Max. allowed time in seconds between two polling cycles. In slave mode, the slave disconnects the link when this time between polls is exceeded.
REF 16	Parameter 1		Protocol specific parameter 1. Range: -2147483648 - +2147483647 If dialing is enabled , this parameter is used for hangup time . This is the time in seconds before hangup in case no data is transmitted. This time is also used for dial-up waiting for answer. Recommended value is 30 seconds.

Table 3-4. Description of MS for Line Characteristics (Continued)

Terminal	Parameter Name	Recommended Value (Min-Max)	Description
REF 17	Parameter 2		Protocol specific parameter 2. Range: -2147483648 - +2147483647 If dialing is enabled, this parameter is used for type of modem which determines the command sequence during auto call. Commands without pause for dial tone after command: 0 = Hayes (without ID 'ATDxxx') 1 = V25.bis (without ID 'CRNxxx') 2 = Hayes (with ID 'ATDxxx;yyy') 3 = V25.bis (with ID 'CRIxxx;yyy') Commands with pause for dial tone after command: 4 = Hayes (without ID 'ATDWxxx') 5 = V25.bis (without ID 'CRN:xxx') 6 = Hayes (with ID 'ATDWxxx;yyy') 7 = V25.bis (with ID 'CRI:xxx;yyy') If 10 is added to command codes 0, 2, 4 and 6, a "P" is added immediately after the 'ATD' to set the modem in pulse dialing mode. If 20 is added to command codes 0, 2, 4 and 6, a "T" is added immediately after the 'ATD' to set the modem in tone dialing mode.
REF 18	Parameter 3		Protocol specific parameter 3. Range: -2147483648 - +2147483647 If dialing is enabled, this parameter is used for switchboard, which is used if telephone connection passes a switchboard. 0 = direct line, 1 = through switchboard by "0", 91 = through switchboard by "9".
REF 19	Parameter 4		Protocol specific parameter 4. Range: -2147483648 - +2147483647 If dialing is enabled, this parameter is used for disconnect time, which is hang-up time in seconds. The time DTR is deactivated during hang-up.

Table 3-4. Description of MS for Line Characteristics (Continued)

Terminal	Parameter Name	Recommended Value (Min-Max)	Description
REF 20	Parameter 5		Protocol specific parameter 5. Range: -2147483648 - +2147483647 If dialing is enabled , this parameter is used for own area number . Used for the ID part in the dial-up command if REF17 = 2, 3, 6 or 7. For a description of the contents, see note below.
REF 21	Parameter 6		Protocol specific parameter 6. Range: -2147483648 - +2147483647 If dialing is enabled , this parameter is used for own phone number , that is, for the ID part in the dial-up command if REF17 = 2, 3, 6 or 7. For a description of the contents, see note below.
REF 22	Parameter 7		Protocol specific parameter 7. Range: -2147483648 - +2147483647
REF 23	Parameter 8		Protocol specific parameter 8. Range: -2147483648 - +2147483647

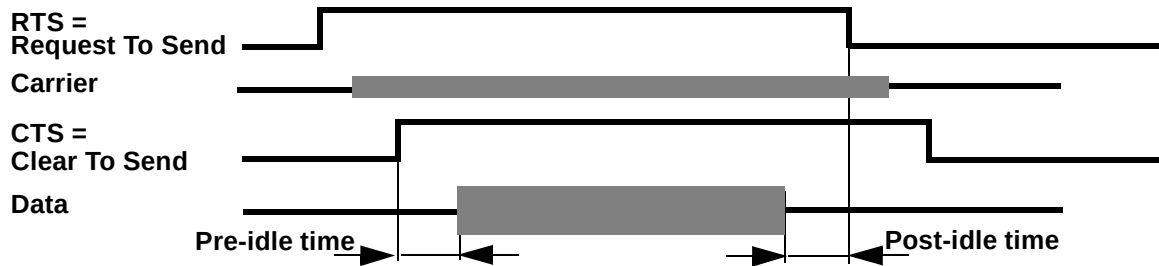


Figure 3-19. Pre-idle Time and Post-idle Time in Half Duplex Mode

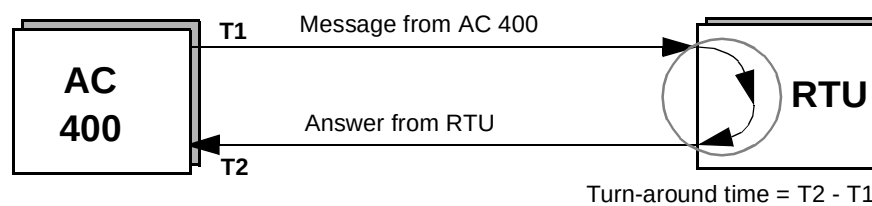


Figure 3-20. Turn-around Time

Network Configuration MS

The node numbers for the RTUs are defined in one MS for each communication port. The Network Configuration MS is read by the MVI protocol during its start-up/restart and at the Advant Ccontroller 400 start-up. If the MVI protocol works as a slave, only the master is to be described by the MS involved. This is done with REF1. **This MS must be blocked at runtime.**

NOTE

The CI535 submodule must always be restarted (see [Section 2.4, Start-up Procedures](#)) if the references in the Network Configuration MS are changed. Sending the Line Characteristics MS to the CI535 submodule for new initiation of the port is **not** enough.

MSn/MSn				MVI Data Set		
				(298.n)		
				Base part		
MSn	_____	1	NAME	VALID	11	—
1	_____	2	ACT	ERR	17	—
2/12	_____	3	IDENT			
0	_____	4	NO_BREC			
0	_____	5	NO_INT			
24	_____	6	NO_INTL			
0	_____	7	NO_REAL			
3	_____	9	USER			
SEND	_____	10	SOURCE			
1	_____	12	BLOCKED			
"submodule number"	_____	13	NET			
-3	_____	14	NODE			
1	_____	15	SCAN_FTR			
YES	_____	18	SORT_REF			
				Value references		
RTU-number for RTU1	_____	8(1)	REF1			
area number for RTU1	_____	8(2)	REF2			
phone number for RTU1	_____	8(3)	REF3			
RTU-number for RTU2	_____	8(4)	REF4			
area number for RTU2	_____	8(5)	REF5			
phone number for RTU2	_____	8(6)	REF6			
RTU-number for RTU3	_____	8(7)	REF7			
area number for RTU3	_____	8(8)	REF8			
phone number for RTU3	_____	8(9)	REF9			
RTU-number for RTU4	_____	8(10)	REF10			
area number for RTU4	_____	8(11)	REF11			
phone number for RTU4	_____	8(12)	REF12			
RTU-number for RTU5	_____	8(13)	REF13			
area number for RTU5	_____	8(14)	REF14			
phone number for RTU5	_____	8(15)	REF15			
RTU-number for RTU6	_____	8(16)	REF16			
area number for RTU6	_____	8(17)	REF17			
phone number for RTU6	_____	8(18)	REF18			
RTU-number for RTU7	_____	8(19)	REF19			
area number for RTU7	_____	8(20)	REF20			
phone number for RTU7	_____	8(21)	REF21			
RTU-number for RTU8	_____	8(22)	REF22			
area number for RTU8	_____	8(23)	REF23			
phone number for RTU8	_____	8(24)	REF24			

Figure 3-21. Example of MVI Data Set for Network Configuration

The parameters REF1 - REF24 are stored in DAT elements referenced by the Network Configuration MS. You can give the DAT elements arbitrary names.

Each RTU requires three DATs: one for the node number and two for the telephone number. If you do not use the dial-up function, you must define the latter two DATs and set them to 0 (zero).

The parameters have the following significance:

Table 3-5. Description of MS for Network Configuration

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT		2 or 12	Identity of the MVI Data Set 2 for port 1 12 for port 2
NO_BREC		0	
NO_INT		0	
NO_INTL		24	Each defined RTU requires 3 Integer Long DATs.
NO_REAL		0	
USER		3	
SOURCE		SEND	
BLOCKED		1	Note: The MS must be blocked. MS requested by the CI535 module at startup.
NET		7 - 11	CI535 submodule number. Record number for the CI535 data base element for the MVI protocol port defines the slave module number. Record No. 1 => submodule = 7 Record No. 2 => submodule = 8 Record No. 3 => submodule = 9 (only valid for AC 450) Record No. 4 => submodule = 10 (only valid for AC 450) Record No. 5 => submodule = 11 (only valid for AC 450)
NODE		-3	Always -3 for Configuration MS
SCAN_FTR		1	Without function when BLOCKED=1
SORT_REF		YES	REF1-24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID			1 after first successful transmission. Reset by user.
ERR			1 if the transmission has failed due to lost contact with destination node or queue full to CI535.

Table 3-5. Description of MS for Network Configuration (Continued)

Terminal	Parameter Name	Recommended Value	Description
REF1 REF4 REF7 REF10 REF13 REF16 REF19 REF22	RTU number: RTU1 RTU2 RTU3 RTU4 RTU5 RTU6 RTU7 RTU8	1 - 99	Node number of: RTU1 RTU2 RTU3 RTU4 RTU5 RTU6 RTU7 RTU8 Note: All the defined node numbers must be unique within the connected controller.
REF2 REF5 REF8 REF11 REF14 REF17 REF20 REF23	Area number: RTU1 RTU2 RTU3 RTU4 RTU5 RTU6 RTU7 RTU8		Area code for: RTU1 in case of telephony. Otherwise = 0. RTU2 in case of telephony. Otherwise = 0. RTU3 in case of telephony. Otherwise = 0. RTU4 in case of telephony. Otherwise = 0. RTU5 in case of telephony. Otherwise = 0. RTU6 in case of telephony. Otherwise = 0. RTU7 in case of telephony. Otherwise = 0. RTU8 in case of telephony. Otherwise = 0.
REF3 REF6 REF9 REF12 REF15 REF18 REF21 REF24	Phone number: RTU1 RTU2 RTU3 RTU4 RTU5 RTU6 RTU7 RTU8		Subscriber number for: RTU1 in case of telephony. Otherwise = 0. RTU2 in case of telephony. Otherwise = 0. RTU3 in case of telephony. Otherwise = 0. RTU4 in case of telephony. Otherwise = 0. RTU5 in case of telephony. Otherwise = 0. RTU6 in case of telephony. Otherwise = 0. RTU7 in case of telephony. Otherwise = 0. RTU8 in case of telephony. Otherwise = 0.

NOTE

The area code description (REF2, REF5, REF8, REF11, REF14, REF17, REF20 and REF23) may also contain a national code. Since national and area codes often start with zeros and since leading zeros are not significant in Integers, precede this part of the description by a digit 1-9. If the description contains only an area code, it must start with a "1".

EXAMPLES

The area number 0921 is entered as 10921.
The area number 1234 is entered as 11234.

If the description includes a national code, the number of digits it consists of is specified by the first digit. A pause is always included for the dial tone after the national code.

EXAMPLES

The national and area code 00946-921 are entered as 500946921.
The national and area code 1234-5678 are entered as 412345678.

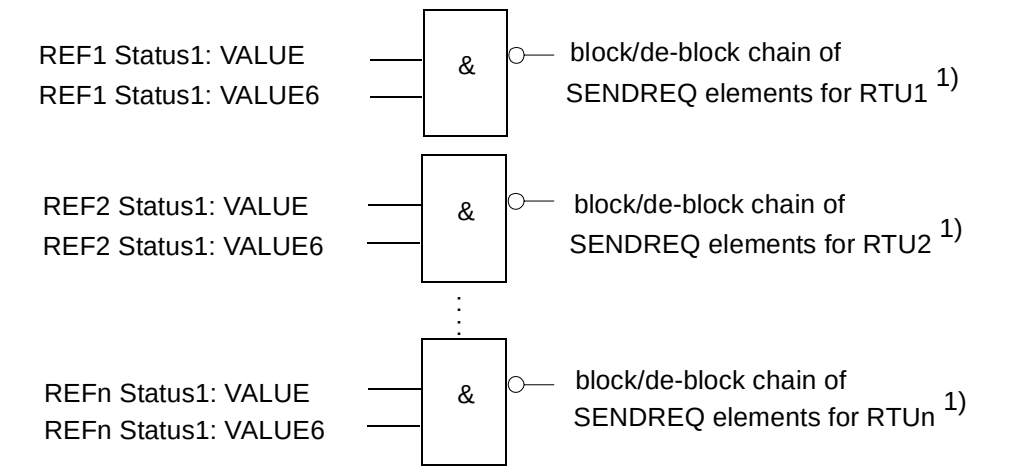
You can enter a maximum of nine digits, including the length code, for the national/area code of the respective phone number.

RTU Status MS

The status of every RTU is stored in an MS for each port. The status word is updated by the MVI protocol and **may only be read by the PC program.**

Use the status information for flow control and error indication to application programs. The flow control is maintained through connecting bit0 (Link status) and bit5 (Ready for Message) in Status1, inverted, to the BLOCK input of all SENDREQ PC elements working against the same RTU (see [Figure 3-22](#)).

The RTU Status MS is updated on every status change and at least every five seconds. This makes it possible for you to use this MS as a Watchdog function for the CI535 submodule, using a time-out supervision on the RTU Status MS 'VALID' flag.



1) See [Figure 3-33](#) for a description of the chain of SENDREQ PC elements.

Figure 3-22. Status Handling for Flow Control

MSn/MSn				MVI Data Set (298.n)	
				Base part	
MSn	_____	1		NAME	VALID 11 —
1	_____	2		ACT	ERR 17 —
3/13	_____	3		IDENT	
8	_____	4		NO_BREC	
0	_____	5		NO_INT	
16	_____	6		NO_INTL	
0	_____	7		NO_REAL	
3	_____	9		USER	
RECEIVE	_____	10		SOURCE	
0	_____	12		BLOCKED	
"submodule number"	_____	13		NET	
-3	_____	14		NODE	
1	_____	15		SCAN_FTR	
YES	_____	18		SORT_REF	
				Value references	
status1 RTU1	_____	8(1)		REF1	
status1 RTU2	_____	8(2)		REF2	
status1 RTU3	_____	8(3)		REF3	
status1 RTU4	_____	8(4)		REF4	
status1 RTU5	_____	8(5)		REF5	
status1 RTU6	_____	8(6)		REF6	
status1 RTU7	_____	8(7)		REF7	
status1 RTU8	_____	8(8)		REF8	
status2 RTU1	_____	8(9)		REF9	
status2 RTU2	_____	8(10)		REF10	
status2 RTU3	_____	8(11)		REF11	
status2 RTU4	_____	8(12)		REF12	
status2 RTU5	_____	8(13)		REF13	
status2 RTU6	_____	8(14)		REF14	
status2 RTU7	_____	8(15)		REF15	
status2 RTU8	_____	8(16)		REF16	
status3 RTU1	_____	8(17)		REF17	
status3 RTU2	_____	8(18)		REF18	
status3 RTU3	_____	8(19)		REF19	
status3 RTU4	_____	8(20)		REF20	
status3 RTU5	_____	8(21)		REF21	
status3 RTU6	_____	8(22)		REF22	
status3 RTU7	_____	8(23)		REF23	
status3 RTU8	_____	8(24)		REF24	

Figure 3-23. MVI Data Set for RTU Status

The parameters REF1 - REF24 are stored in DAT elements referenced from this Status MS. You can give the DAT elements arbitrary names.

The parameters have the following significance:

Table 3-6. Description of MS for RTU Status

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT		3 or 13	Identity of the MVI Data Set 3 for port 1 13 for port 2
NO_BREC		8	Number of Boolean DAT references
NO_INT		0	
NO_INTL		16	Number of Integer Long DAT references
NO_REAL		0	
USER		3	
SOURCE		RECEIVE	
BLOCKED		0	Note: The MS must not be blocked.
NET		7 - 11	CI535 submodule number. Record number for the CI535 data base element for the MVI protocol port defines the slave module number. Record No. 1 => submodule = 7 Record No. 2 => submodule = 8 Record No. 3 => submodule = 9 (only valid for AC 450) Record No. 4 => submodule = 10 (only valid for AC 450) Record No. 5 => submodule = 11 (only valid for AC 450)
NODE		-3	Always -3 for Configuration MS.
SCAN_FTR		1	Used for reset of the VALID flag. VALID is set to 0 when the Time = 3*SCAN_FTR * MS_SCANTIME has passed. Default value for MS_SCANTIME = 2 s.
SORT_REF		YES	REF1 - 24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID			1 when the MS is updated. See also SCAN_FTR above.
ERR			1 if wrong data size is received

Table 3-6. Description of MS for RTU Status (Continued)

Terminal	Parameter Name	Recommended Value	Description
REF1	Status1 RTU1		First status word for RTU1
REF2	Status1 RTU2		First status word for RTU2
REF3	Status1 RTU3		First status word for RTU3
REF4	Status1 RTU4		First status word for RTU4
REF5	Status1 RTU5		First status word for RTU5
REF6	Status1 RTU6		First status word for RTU6
REF7	Status1 RTU7		First status word for RTU7
REF8	Status1 RTU8		First status word for RTU8
REF9	Status2 RTU1		Second status word for RTU1
REF10	Status2 RTU2		Second status word for RTU2
REF11	Status2 RTU3		Second status word for RTU3
REF12	Status2 RTU4		Second status word for RTU4
REF13	Status2 RTU5		Second status word for RTU5
REF14	Status2 RTU6		Second status word for RTU6
REF15	Status2 RTU7		Second status word for RTU7
REF16	Status2 RTU8		Second status word for RTU8
REF17	Status3 RTU1		Third status word for RTU1
REF18	Status3 RTU2		Third status word for RTU2
REF19	Status3 RTU3		Third status word for RTU3
REF20	Status3 RTU4		Third status word for RTU4
REF21	Status3 RTU5		Third status word for RTU5
REF22	Status3 RTU6		Third status word for RTU6
REF23	Status3 RTU7		Third status word for RTU7
REF24	Status3 RTU8		Third status word for RTU8

NOTE

You **must** build this MS with 24 DATs, according to the description above.

Status1 RTU1 - RTU8 (Boolean)

Contains information required for flow control, among other things. See [Table 3-7](#), below.

Table 3-7. Status1 RTU1 - RTU8 (Boolean)

Status1	Description	Valid modes
VALUE (bit0)	Link status The corresponding RTU reachable, see Figure 3-22 . Set to '1' when the remote node is reachable. Set to '0' when contact with remote node is lost (including restart of CI535).	Slave and Master
VALUE2 (bit1)	Reserved	
VALUE3 (bit2)	Reserved	
VALUE4 (bit3)	Illegal MS number received Set to '1' when illegal MS number is received from an RTU or from an AMPL application command. Set to '0' when receiving valid MS or command.	Slave and Master
VALUE5 (bit4)	Illegal function code received Set to '1' when illegal functions code is received from an RTU. Set to '0' when receiving valid MS.	Slave
VALUE6 (bit5)	Ready for Message Set to '1' when no command is pending, from the application to the RTU. Set to '0' when a command is pending, from the application to the RTU. Used by the PC program for flow control, see Figure 3-22 .	Master
VALUE7 (bit6)	Reserved	
VALUE8 (bit7)	Reserved	
VALUE9 (bit8) to VALUE32 (bit31)	User defined	

Status2 RTU1 - RTU8 (Integer Long)

If Status1 VALUE4 = 1 or VALUE8 = 1, for example, address error, the Status2 holds the address that caused the error. Status2 is reset at restart of the CI535.

Status3 RTU1 - RTU8 (Integer Long)

If Status1 VALUE5 = 1 or VALUE7 = 1, for example, illegal function code, the Status3 holds the function code that caused the error. Status3 is reset at restart of the CI535.

RTU Diagnostics Information MS

The diagnostics information of the UART Controller and every RTU is stored in an MS for each port. The diagnostics information is updated by the MVI protocol and **may only be read by the PC program.**

		MSn/MSn	
		MVI Data Set (298.n)	
		Base part	
MSn	_____	1	NAME VALID 11 _____
1	_____	2	ACT ERR 17 _____
254/255	_____	3	IDENT
1	_____	4	NO_BREC
0	_____	5	NO_INT
18	_____	6	NO_INTL
0	_____	7	NO_REAL
3	_____	9	USER
RECEIVE	_____	10	SOURCE
0	_____	12	BLOCKED
"submodule number"	_____	13	NET
-3	_____	14	NODE
1	_____	15	SCAN_FTR
YES	_____	18	SORT_REF
		Value references	
controller status	_____	8(1)	REF1
controller error	_____	8(2)	REF2
parity error counter RTU1	_____	8(3)	REF3
other errors counter RTU1	_____	8(4)	REF4
parity error counter RTU2	_____	8(5)	REF5
other errors counter RTU2	_____	8(6)	REF6
parity error counter RTU3	_____	8(7)	REF7
other errors counter RTU3	_____	8(8)	REF8
parity error counter RTU4	_____	8(9)	REF9
other errors counter RTU4	_____	8(10)	REF10
parity error counter RTU5	_____	8(11)	REF11
other errors counter RTU5	_____	8(12)	REF12
parity error counter RTU6	_____	8(13)	REF13
other errors counter RTU6	_____	8(14)	REF14
parity error counter RTU7	_____	8(15)	REF15
other errors counter RTU7	_____	8(16)	REF16
parity error counter RTU8	_____	8(17)	REF17
other errors counter RTU8	_____	8(18)	REF18

Figure 3-24. MVI Data Set for Diagnostics Information

Table 3-8. Description of MS for Diagnostics Information

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT		254 or 255	Identity of the MVI Data Set 254 for port 1 255 for port 2
NO_BREC		1	Number of Boolean DAT references
NO_INT		0	
NO_INTL		18	Number of Integer Long DAT references
NO_REAL		0	
USER		3	
SOURCE		RECEIVE	
BLOCKED		0	Note: The MS must not be blocked.
NET		7-11	CI535 submodule number. Record number for the CI535 data base element for the MVI Demo Protocol port defines the slave module number. Record No. 1 => submodule = 7 Record No. 2 => submodule = 8 Record No. 3 => submodule = 9 (only valid for AC 450) Record No. 4 => submodule = 10 (only valid for AC 450) Record No. 5 => submodule = 11(only valid for AC 450)
NODE		-3	Always -3 for Configuration MS.
SCAN_FTR		1	Used for reset of the VALID flag. VALID is set to 0 when the Time = 3*SCAN_FTR * MS_SCANTIME has passed. Default value for MS_SCANTIME = 2 s.
SORT_REF		YES	REF1 - 24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID			1 when the MS is updated. See also SCAN_FTR above.
ERR			1 if wrong data size is received

Table 3-8. Description of MS for Diagnostics Information (Continued)

Terminal	Parameter Name	Recommended Value	Description
REF1	Controller status		Current status of UART modem signals
REF2	Controller error		UART Controller errors counter
REF3	Parity error RTU1		Parity error word for RTU1
REF4	Other errors RTU1		Other errors counter for RTU1
REF5	Parity error RTU2		Parity error word for RTU2
REF6	Other errors RTU2		Other errors counter for RTU2
REF7	Parity error RTU3		Parity error word for RTU3
REF8	Other errors RTU3		Other errors counter for RTU3
REF9	Parity error RTU4		Parity error word for RTU4
REF10	Other errors RTU4		Other errors counter for RTU4
REF11	Parity error RTU5		Parity error word for RTU5
REF12	Other errors RTU5		Other errors counter for RTU5
REF13	Parity error RTU6		Parity error word for RTU6
REF14	Other errors RTU6		Other errors counter for RTU6
REF15	Parity error RTU7		Parity error word for RTU7
REF16	Other errors RTU7		Other errors counter for RTU7
REF17	Parity error RTU8		Parity error word for RTU8
REF18	Other errors RTU8		Other errors counter for RTU8

Table 3-9. Controller Status (Boolean)

Status	Description
VALUE (bit0)	Current status of Data Set Ready (DSR)
VALUE2 (bit1)	Current status of Clear To Send (CTS)
VALUE3 (bit2)	Current status of Data Carrier Delay (DCD)
VALUE4 (bit3)	Current status of Ring Indicator (RI)
VALUE9 (bit8)	If set to 1, a hardware error was detected

Register Address MS

A translation between the AC 400 MS identities and the MVI protocol register addresses must be made, since the addressing of registers differs between ABB Master and the MVI protocol.

For this purpose, a cross-reference table is used for the MVI protocol. This table is defined by 1 to 4 Register Address MS that must be built. These Register Address MS must contain the MVI protocol register addresses.

Each DAT reference in the Register Address MS corresponds to a Data MS. The DAT elements connected to the Register Address MS hold a MVI protocol register address corresponding to the start of the data (first DAT) in the corresponding Data MS. See [Section 3.6.7.3, Data MS for Data Transfer](#). (Only the Data MS used for data exchange must be built.)

The same start register addresses apply to both **sending** Data MS (IDENT=1-96) and **receiving** Data MS (IDENT=101-196). See also [Figure 3-25](#).

NOTE

In order to update only a part of a MS, there must always be a sending MS (IDENT = 1-96) corresponding to a receiving MS (IDENT = 101-196).

The sending and receiving Data MS are connected to the same DAT elements.

When the CI535 submodule receives data from the RTU, the sending Data MS is requested from AC 400. The received data from the RTU is mapped into the data from the sending Data MS before it is sent back to the receiving Data MS in the AC 400.

The First Register Address MS (IDENT=4 for port 1 and 14 for port 2) holds the MVI protocol start register address for:

- Data MS with IDENT= 1 and 101 in REF1
- Data MS with IDENT= 2 and 102 in REF2
-
- Data MS with IDENT= 24 and 124 in REF24

The Second Register Address MS (IDENT=5 for port 1 and 15 for port 2) holds the MVI protocol start register address for:

- Data MS with IDENT= 25 and 125 in REF1
- Data MS with IDENT= 26 and 126 in REF2
-
-
- Data MS with IDENT= 48 and 148 in REF24

The Third Register Address MS (IDENT=6 for port 1 and 16 for port 2) holds the MVI protocol start register address for:

- Data MS with IDENT= 1 and 101 in REF1
- Data MS with IDENT= 2 and 102 in REF2
-
-
- Data MS with IDENT= 72 and 172 in REF24

The Fourth Register Address MS (IDENT = 7 for port 1 and 17 for port 2) holds the MVI protocol start register address for:

- Data MS with IDENT = 73 and 173 in REF1
- Data MS with IDENT = 74 and 174 in REF2
-
-
- Data MS with IDENT= 96 and 196 in REF24

The DAT elements in the **Data MS** must be connected as follows:

- Data MS with IDENT=1 reference the same DAT element as the Data MS with IDENT=101.
- Data MS with IDENT=2 reference the same DAT element as the Data MS with IDENT=102.
- Data MS with IDENT=3 reference...
- And so forth.

The same register address must **not** exist in two different Data MS. These Register Address MS must **always be blocked**. They are downloaded to the CI535 submodule at start-up/restart.

NOTE

If the start addresses for the MVI Data Sets are changed, the submodule must be restarted to re-initialize the cross-reference table. If, on the other hand, the size of a Data MS is changed within the allowed limits, no restart is necessary.
See [Section 3.6.7.3, Data MS for Data Transfer](#).

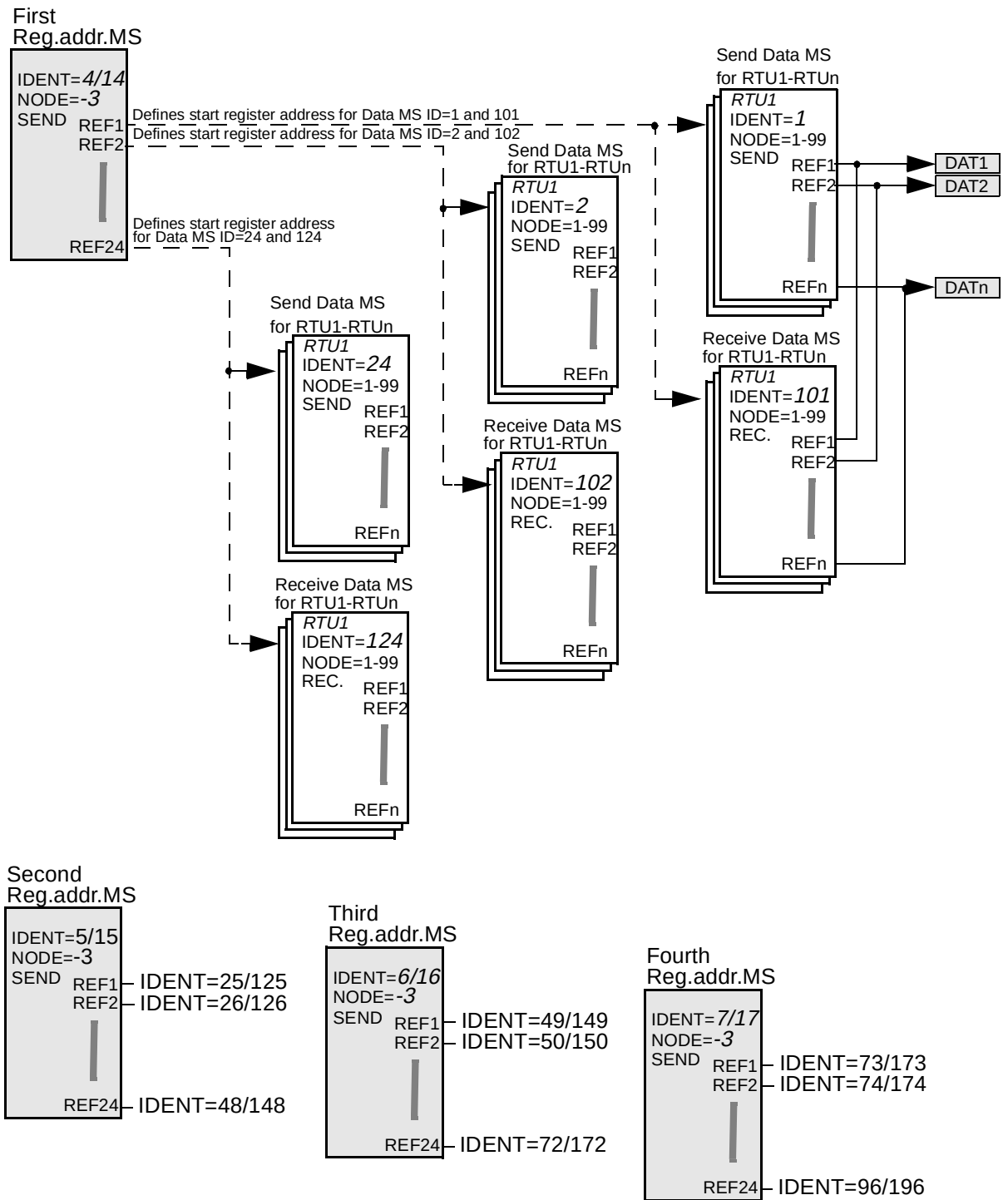


Figure 3-25. Register Address MS - Data MS Relationship

Each of the Register Address MS must be built with 24 DATs. The start addresses for the Data MS must be written consecutive to the Register Address MS. This means that no holes are permitted in the Register Address MS. The end of Data MS is marked by writing -1 (minus one) into the first unused DAT and into the rest of the DAT's of the Register Address MS.

Between one and four Register Address MS must be built, depending on how many Data MS are used.

When Boolean registers are used in the Data MS, each DAT(B) contains the equivalent of 32 MVI protocol registers. Every integer DAT(I), integer long DAT(IL) and real DAT(R) element corresponds to one MVI protocol register.

Example

If MS IDENT=1 (or IDENT=101) is set at 0001 in the Register Address MS, and MS with IDENT=1 (or IDENT=101) is configured for 20 DATs, it contains:

20 DAT(B) * 32 = 640 Boolean registers,

where VALUE in the first DAT(B) corresponds to register address 0001, and VALUE2 corresponds to register address 0002. VALUE32 in the last DAT(B) corresponds to register address 0640. In this case, MS IDENT=2 (or IDENT=102) must be set at 0641 or higher in the Register Address MS (if that Data MS is used) to avoid overlapping.

If MS IDENT=17 (or IDENT=117) is set at 3001 in the Register Address MS, and the former is configured for 20 DATs, it contains:

20 DAT(I) integer (2 bytes) registers.

Then, MS IDENT=18 (or IDENT=118) must be set at 3021 or higher in the Register Address MS (if that Data MS is used) to avoid overlapping.

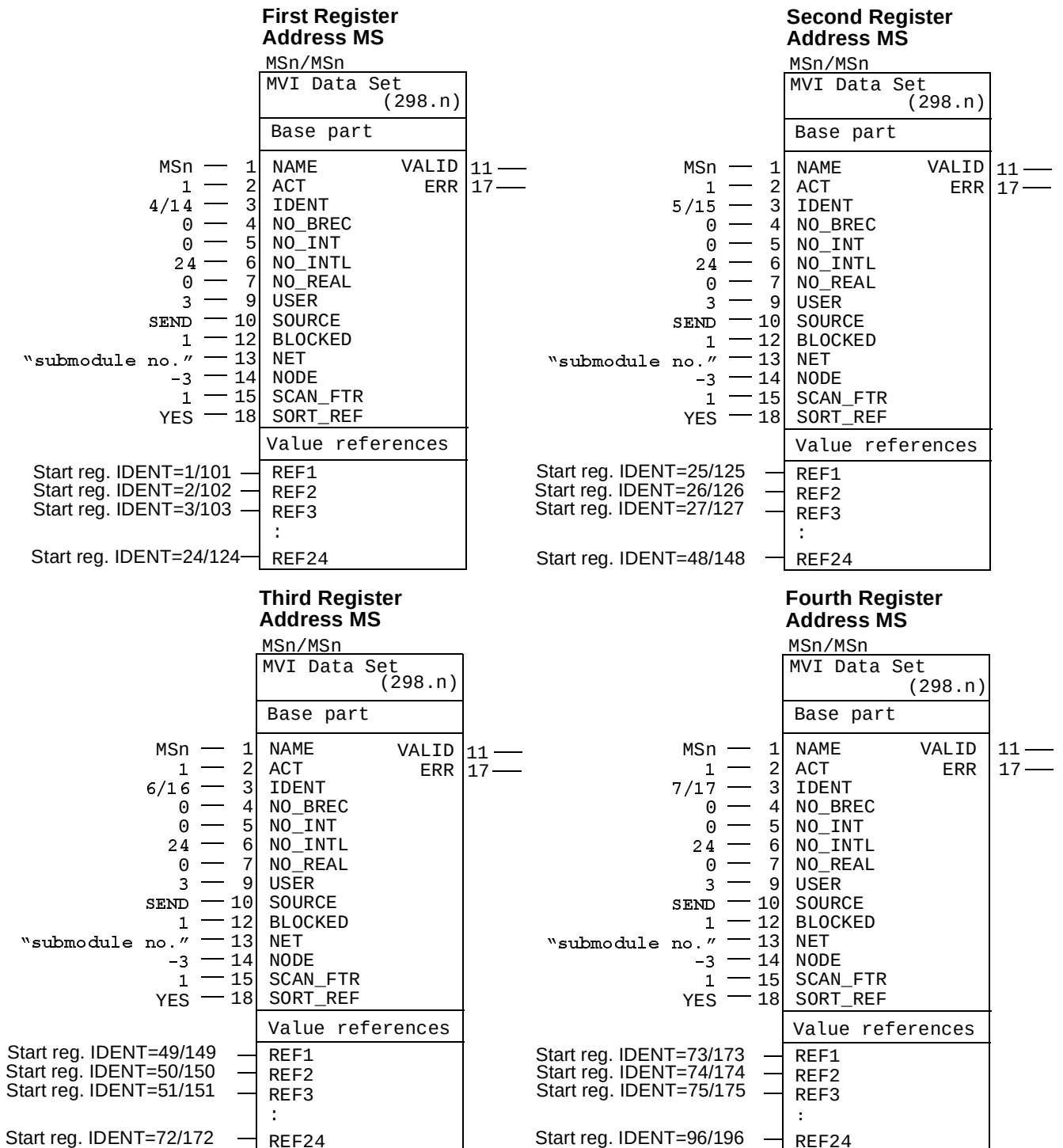


Figure 3-26. MVI Data Sets for Register Addresses

The parameters REF1 - REF24 are stored in DAT elements referred to by this MVI Data Set. You can give the DAT elements arbitrary names.

The parameters have the following function:

Table 3-10. Description of Register Address MS

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT		Port1: 4 - 7 Port2: 14 - 17	Identity of the MVI Data Set -First Register Address MS: Port 1: IDENT = 4 Port 2: IDENT = 14 -Second Register Address MS: Port 1: IDENT = 5 Port 2: IDENT = 15 -Third Register Address MS: Port 1: IDENT = 6 Port 2: IDENT = 16 -Fourth Register Address MS: Port 1: IDENT = 7 Port 2: IDENT = 17
NO_BREC		0	
NO_INT		0	
NO_INTL		24	Number of Integer Long DAT references.
NO_REAL		0	
USER		3	
SOURCE		SEND	
BLOCKED		1	Note: The MS must be blocked. MS requested by the CI535 module at start-up.
NET		7-11	CI535 submodule number. Record number for the CI535 data base element for the MVI protocol port defines the slave module number. Record No. 1 => submodule = 7 Record No. 2 => submodule = 8 Record No. 3 => submodule = 9 (only valid for AC 450) Record No. 4 => submodule = 10 (only valid for AC 450) Record No. 5 => submodule = 11(only valid for AC 450)

Table 3-10. Description of Register Address MS (Continued)

Terminal	Parameter Name	Recommended Value	Description
NODE		-3	
SCAN_FTR		1	Without function when BLOCKED=1.
VALID		1	1 after first successful transmission. Reset by user.
ERR			1 if the transmission has failed due to lost contact with destination node or queue full to CI535.
REF1 - REF24			Start register address for corresponding to: Sending Data MS with IDENT =1 - 96 and Receiving Data MS with IDENT = 101 - 196. See Table 3-11 for relation between REF in Register Address MS and IDENT for Data MS.

Table 3-11. Relation between REF in Reg. Addr. MS REF & IDENT for Data MS

DAT Reference in Register Address MS	Data MS IDENT in First Register Address MS (Send/Receive)	Data MS IDENT in Second Register Address MS (Send/Receive)	Data MS IDENT in Third Register Address MS (Send/Receive)	Data MS IDENT in Fourth Register Address MS (Send/Receive)
REF1	1/101	25/125	49/149	73/173
REF2	2/102	26/126	50/150	74/174
REF3	3/103	27/127	51/151	75/175
REF4	4/104	28/128	52/152	76/176
REF5	5/105	29/129	53/153	77/177
REF6	6/106	30/130	54/154	78/178
REF7	7/107	31/131	55/155	79/179
REF8	8/108	32/132	56/156	80/180
REF9	9/109	33/133	57/157	81/181
REF10	10/110	34/134	58/158	82/182
REF11	11/111	35/135	59/159	83/183
REF12	12/112	36/136	60/160	84/184
REF13	13/113	37/137	61/161	85/185
REF14	14/114	38/138	62/162	86/186
REF15	15/115	39/139	63/163	87/187
REF16	16/116	40/140	64/164	88/188
REF17	17/117	41/141	65/165	89/189
REF18	18/118	42/142	66/166	90/190
REF19	19/119	43/143	67/167	91/191
REF20	20/120	44/144	68/168	92/192
REF21	21/121	45/145	69/169	93/193
REF22	22/122	46/146	70/170	94/194
REF23	23/123	47/147	71/171	95/195
REF24	24/124	48/148	72/172	96/196

Example of Register Addresses

The minimum number of Register Address MS depends on the number of Data MS to be used.

Table 3-12. Minimum Number of Register Address MS

Number of Data MS	Number of Register Address MS
1 - 23	1
24 - 47	2
48 - 71	3
72 - 96	4

This example describes a configuration that uses 14 Data MS. The first Register Address MS (IDENT = 4 or 14) could have the following configuration:

Table 3-13. Example of Register Addresses

DAT reference in First Register Address MS	DAT Value	Data MS IDENT (Send/Receive)
REF1	1	1/101
REF2	641	2/102
REF3	100	3/103
REF4	107	4/104
REF5	300	5/105
REF6	929	6/106
REF7	400	7/107
REF8	425	8/108
REF9	410	9/109
REF10	450	10/110
REF11	415	11/111
REF12	200	12/112
REF13	250	13/113
REF14	110	14/114
REF15	-1 (See Note Below)	15/115

Table 3-13. Example of Register Addresses (Continued)

DAT reference in First Register Address MS	DAT Value	Data MS IDENT (Send/Receive)
REF16	-1	16/116
REF17	-1	17/117
REF18	-1	18/118
REF19	-1	19/119
REF20	-1	20/120
REF21	-1	21/121
REF22	-1	22/122
REF23	-1	23/123
REF24	-1	24/124

NOTE

The DAT value for the last reference in the Register Address MS must always end with “-1” (except when all 96 Data MS are used).

The other three Register Address MS (IDENT = 5, 6, 7 or 15, 16, 17) do not have to be built in this example.

The MVI protocol register addresses do not have to be in order in the Register Address MS.

No holes in the Register Address MS are permitted. For example:

~~REF1 (MS with IDENT=1/102) = 1
REF2 (MS with IDENT=2/102) = -1
REF3 (MS with IDENT=3/103) = 10001~~

This configuration is not permitted. The Data MS (IDENT=3/103) with start address 10001 are not accessible in this example.

3.6.7.2 Command MS used in Master Mode

This section describes the Command MS used for control of the data transfer in master mode. The Command MS are only defined and used if the CI535 submodule is used as a master on the MVI protocol link. All Command MS used must be included in the send sequence according to [Section 3.6.7.4, Type Circuits for MVI Protocol Data Flow Control](#).

These Command MS are used by the application program to activate read and write commands from CI535 to the corresponding RTU slave.

Control Commands

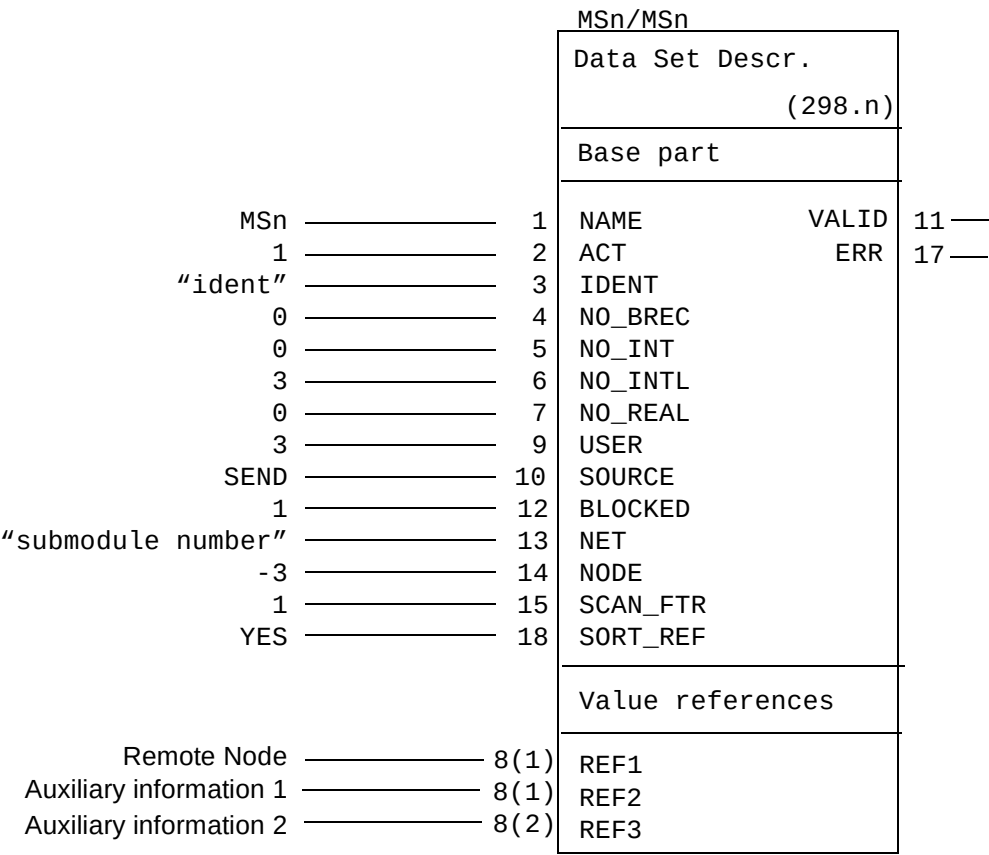


Figure 3-27. Control Command MS

Table 3-14. Description of MS for Control Commands

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT			The identity of the MVI Data Set identifies the command code: 101= Restart Handler Port 1 111 = Restart Handler Port 2 102 = Reset Diagnostics Port 1 112 = Reset Diagnostics Port 2 103 = Read Diagnostics Port 1 113 = Read Diagnostics Port 2 104 = Dial Station Port 1 114 = Dial Station Port 2 105 = Hang Up Line Port 1 115 = Hang Up Line Port 2
NO_BRE C		0	
NO_INT		0	
NO_INTL		3	Number of Integer Long DAT references
NO_REAL		0	
USER		3	
SOURCE		SEND	
BLOCKE D		1	Note: The MS must be blocked.
NET		7 - 11	CI535 submodule number. Record number for the CI535 data base element for the MVI protocol port defines the slave module number. Record No. 1 => submodule = 7 Record No. 2 => submodule = 8 Record No. 3 => submodule = 9 (only valid for AC 450) Record No. 4 => submodule = 10 (only valid for AC 450) Record No. 5 => submodule = 11 (only valid for AC 450)
NODE		-3	Always -3 for Configuration MS
SCAN_F TR		1	Without function when BLOCKED=1

Table 3-14. Description of MS for Control Commands (Continued)

Terminal	Parameter Name	Recommended Value	Description
SORT_REF		YES	REF1-24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID		1	1 after first successful transmission. Reset by user.
ERR			1 if the transmission has failed due to lost contact with destination node or queue full to CI535
REF 1	Remote Node		Usage depends on the command: Ident 101, 111 = Restart Handler: Not used Ident 102, 112 = Reset Diagnostics: Not used Ident 103, 113 = Read Diagnostics: Not used Ident 104, 114 = Dial Station: Node number of remote node Ident 105, 115 = Hang Up Line: Node number of remote node
REF2	Auxiliary information 1		Usage depends on the command: Ident 101, 111 = Restart Handler: Not used Ident 102, 112 = Reset Diagnostics: Not used Ident 103, 113 = Read Diagnostics: Not used Ident 104, 114 = Dial Station: Number of re-dials Ident 105, 115 = Hang Up Line: Not used
REF3	Auxiliary information 2		Usage depends on the command: Ident 101, 111 = Restart Handler: Not used Ident 102, 112 = Reset Diagnostics: Not used Ident 103, 113 = Read Diagnostics: Not used Ident 104, 114 = Dial Station: User defined Ident 105, 115 = Hang Up Line: Not used

Read Commands

To update a value in AC 400, a Read Command must be issued to the corresponding slave unit. A number of Read Command MS are used for this purpose (IDENT=201-210).

The Read Command MS contain four DATs:

- The first DAT contains the start register address to read.
- The second DAT contains the number of registers (values) to read.
- The third DAT contains Auxiliary information 1.
- The fourth DAT contains Auxiliary information 2.

The data returned by the RTU is received by a receiving Data MS with the corresponding register address and IDENT=101-196 according to the Register Address MS with IDENT=4, 5, 6, 7 (port 1) or 14, 15, 16, 17 (port 2).

No Read Command MS are to be created when CI535 operates as a slave on the MVI protocol link, since all data transfer is handled with commands from the master only.

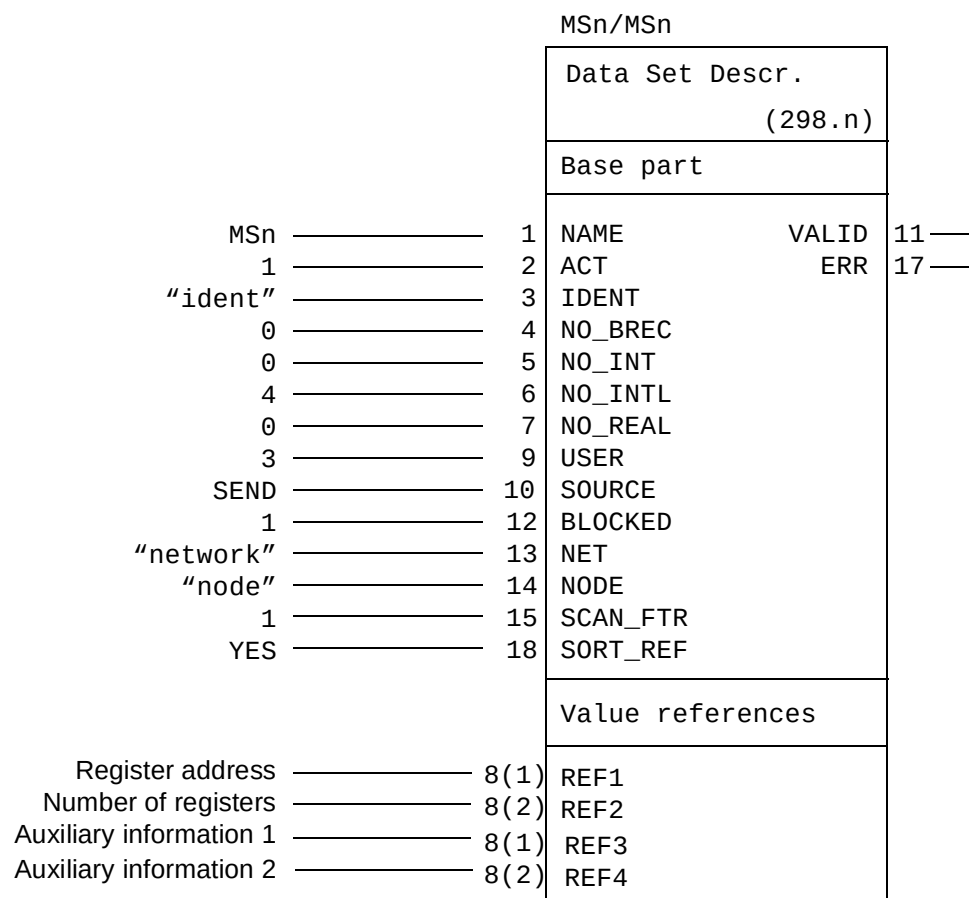


Figure 3-28. Read Command MS

The parameters REF1 - REF4 are stored in DAT elements referred to by this MVI Data Set. You can give the DAT elements arbitrary names.

The parameters have the following function:

Table 3-15. Description of MS for Read Command

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT			The identity of the MVI Data Set identifies the command code: 201 = Read Single Int16 202 = Read Multiple Int16 203 = Read Single Int32 204 = Read Multiple Int32 205 = Read Single Boolean 206 = Read Multiple Boolean 207 = Read Single Float 208 = Read Multiple Float 209 = Read String 210 = Read Mix32
NO_BREC		0	
NO_INT		0	
NO_INTL		4	Number of Integer Long DAT references
NO_REAL		0	
USER		3	
SOURCE		SEND	
BLOCKED		1	Note: The MS must be blocked.
NET		1 - 9	Network number of the MVI Protocol bus. Defined in the CI535 data base element for the CI535 submodule. See Section 2.2.1, Hardware .
NODE		1 - 99	RTU number (node) of the receiving RTU. Valid node numbers must be defined in the Network Configuration MS. NOTE: All the defined node numbers must be unique within the connected controller.
SCAN_FTR		1	Without function when BLOCKED=1
SORT_REF		YES	REF1 - 24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID		1	1 after first successful transmission. Reset by user.

Table 3-15. Description of MS for Read Command (Continued)

Terminal	Parameter Name	Recommended Value	Description
ERR			1 if the transmission has failed due to lost contact with destination node or queue full to CI535
REF 1	Register address		The first register address to be read. This register address must be found in a MS defined in one of the Register Address MS, with IDENT = 4, 5, 6, 7 (port 1) or 14, 15, 16, 17 (port 2), in the master or the slave. The address is written by the PC program before the MS is sent.
REF2	Number of registers		The number of registers to be read. The PC program enters the number of registers before the Command MS is sent.
REF3	Auxiliary information 1		Protocol dependent information. This information can be used in the Protocol Handler to distinguish between several protocol function codes where the command code (ident) is identical.
REF4	Auxiliary information 2		Protocol dependent information. This information can be used in the Protocol Handler to distinguish between several protocol function codes where the command code (ident) is identical.

Write Commands

To update a value in a slave, a Write Command must be issued to the corresponding slave unit. A number of Command MS are used for this purpose (IDENT=221 -230).

The Write Command MS contain four DATs:

- The first DAT contains the start register address to write.
- The second DAT contains the number of registers (values) to write
- The third DAT contains Auxiliary information 1.
- The fourth DAT contains Auxiliary information 2.

The data to be sent is fetched by the MVI protocol from a Data MS (IDENT=1-96) containing the defined registers according to the definition entered into the Register Address MS with IDENT = 4, 5, 6, 7 (port 1) or 14, 15, 16, 17 (port 2).

No Write Command MS is to be created when the MVI protocol operates as a slave on the MVI protocol link, since all data transfer is handled with commands from the master only.

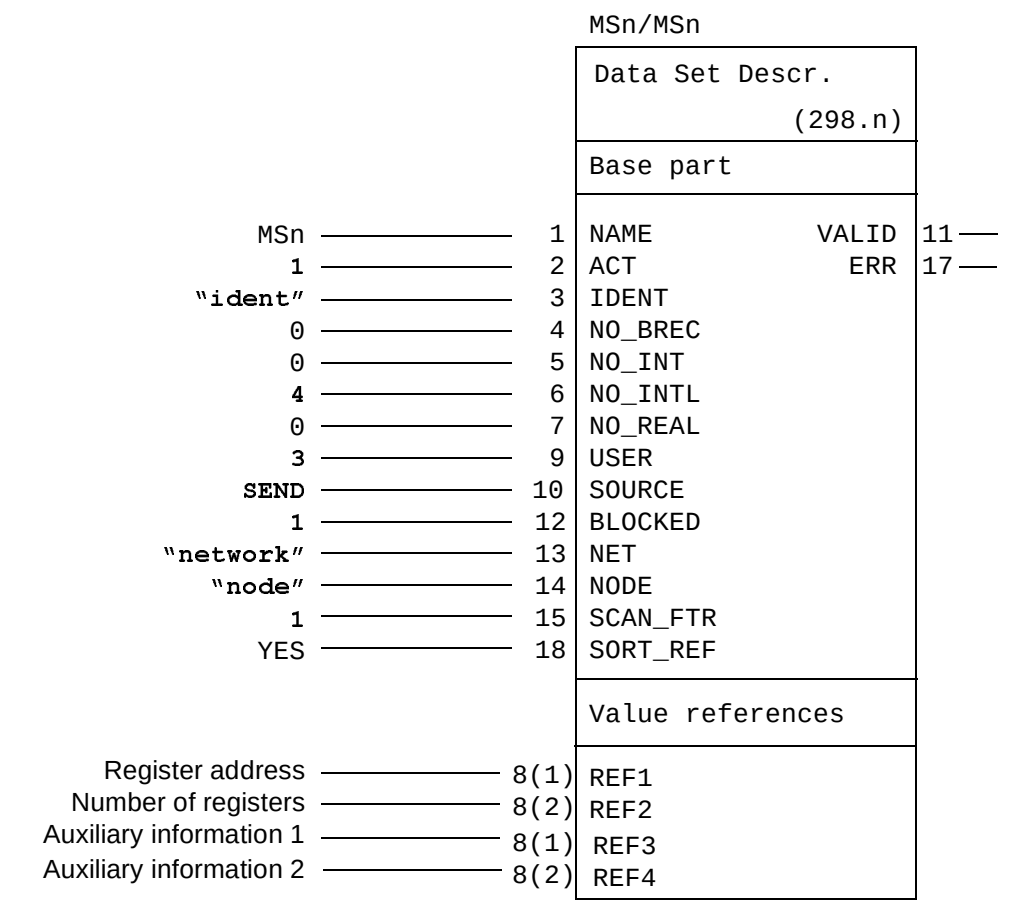


Figure 3-29. Write Command MS

The parameters REF1 - REF4 are stored in DAT elements referred to by this MVI Data Set. You can give the DAT elements arbitrary names.

The parameters have the following function:

Table 3-16. Description of MS for Write Command

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT			The identity of the MVI Data Set identifies the command code: 221 = Write Single Int16 222 = Write Multiple Int16 223 = Write Single Int32 224 = Write Multiple Int32 225 = Write Single Boolean 226 = Write Multiple Boolean 227 = Write Single Float 228 = Write Multiple Float 229 = Write String 230 = Write Mix32
NO_BREC		0	
NO_INT		0	
NO_INTL		4	Number of Integer Long DAT references
NO_REAL		0	
USER		3	
SOURCE		SEND	
BLOCKED		1	Note: The MS must be blocked.
NET		1 - 9	Network number of the MVI protocol bus. Defined in the CI535 data base element for the CI535 submodule. See Section 2.2.1, Hardware .
NODE		1 - 99	RTU number (node) of the receiving RTU. Valid node numbers must be defined in the Network Configuration MS. Note: All the defined node numbers must be unique within the connected controller.
SCAN_FTR		1	Without function when BLOCKED=1
SORT_REF		YES	REF1 - 24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID		1	1 after first successful transmission. Reset by user.

Table 3-16. Description of MS for Write Command (Continued)

Terminal	Parameter Name	Recommended Value	Description
ERR			1 if the transmission has failed due to lost contact with destination node or queue full to CI535
REF 1	Register address		The first register address to be written. This register address must be found in a MS defined in one of the Register Address MS, with IDENT = 4, 5, 6, 7 (port 1) or 14, 15, 16, 17 (port 2), in the master or the slave. The address is written by the PC program before the MS is sent.
REF2	Number of registers		The number of registers to be written. The PC program enters the number of registers before the Command MS is sent.
REF3	Auxiliary information 1		Protocol dependent information. This information can be used in the Protocol Handler to distinguish between several protocol function codes where the command code (ident) is identical.
REF4	Auxiliary information 2		Protocol dependent information. This information can be used in the Protocol Handler to distinguish between several protocol function codes where the command code (ident) is identical.

3.6.7.3 Data MS for Data Transfer

Data Transmission to an RTU

MS for transmission of data to an RTU are defined in the same way as normal sending Data Set. The difference is that the MS used for data transmission on MVI protocol must be blocked (BLOCKED = 1) and USER = 3.

Master Mode

The transmission of the MS is controlled by the Write Command MS (IDENT = 205, 206, 215 or 216) together with the CI535 submodule.

The transmission is controlled by a PC program (see [Section 3.6.7.4, Type Circuits for MVI Protocol Data Flow Control](#)) which sends a Write Command MS to the CI535 submodule. The CI535 submodule requests the sending Data MS containing the addressed registers and transmits them to the addressed RTU.

Slave Mode

The transmission of the Data MS is controlled by a Read Command from the MVI protocol master function code 1-4 together with the CI535 submodule.

The CI535 submodule requests the sending Data MS containing the addressed registers and returns the defined registers to the master.

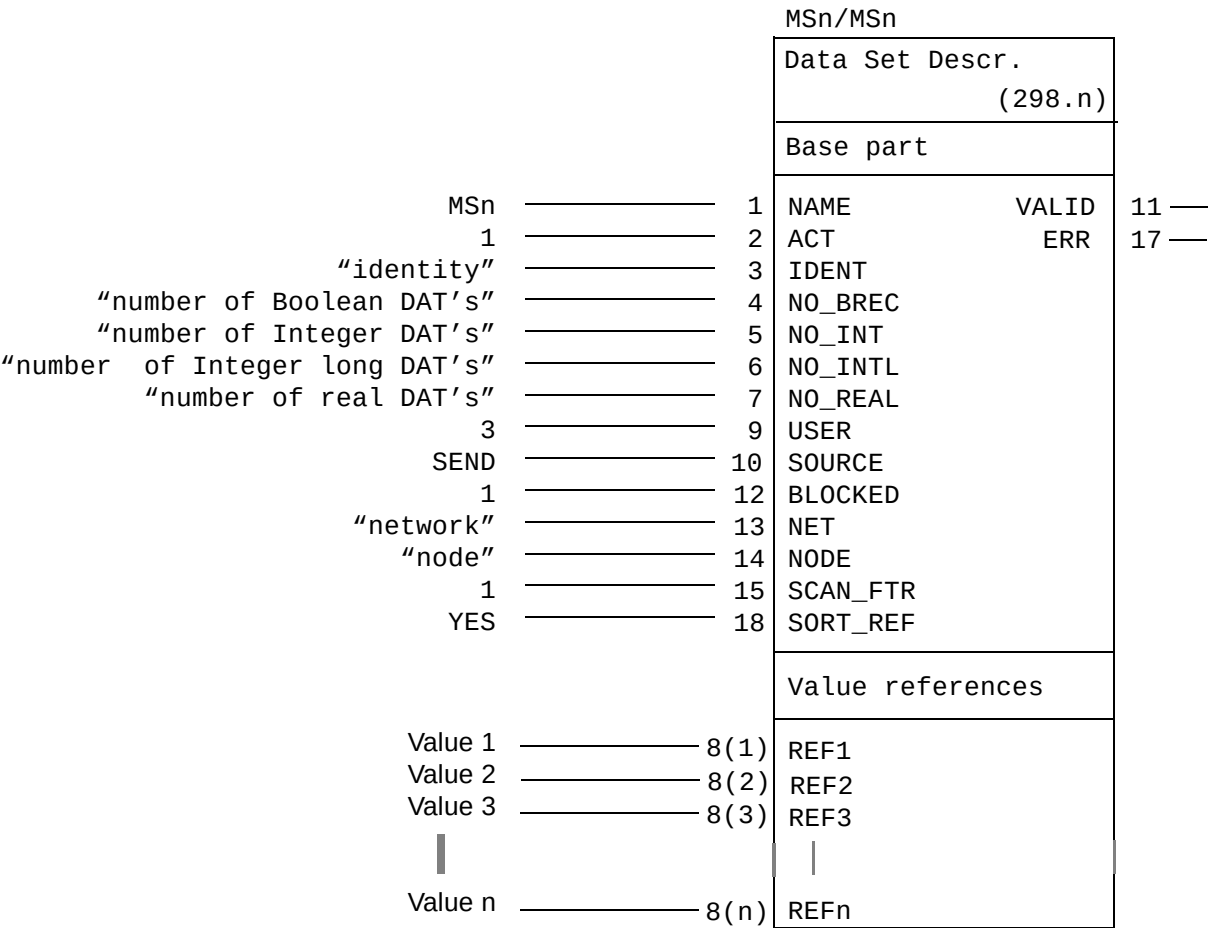


Figure 3-30. MVI Data Set for Transmission of Data

The parameters REF1 - REF24 are stored in DAT elements referred to by this MVI Data Set. You can give the DAT elements arbitrary names.

The parameters have the following function:

Table 3-17. Description of MS for Transmission of Data

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT			The identity of the MVI Data Set (1-96)
NO_BREC		0 - 24	Note: Only one type of REF could be defined.
NO_INT		0 - 24	Note: Only one type of REF could be defined.
NO_INTL		0 - 24	Note: Only one type of REF could be defined.
NO_REAL		0 - 24	Note: Only one type of REF could be defined.
USER		3	
SOURCE		SEND	
BLOCKED		1	Note: The MS must be blocked.
NET		1 - 9	Network number of the MVI protocol bus. Defined in the CI535 data base element for the CI535 submodule. See Section 2.2.1, Hardware .
NODE		1 - 99	RTU number (node) of the receiving RTU. Valid node numbers must be defined in the Network Configuration MS. Note: All the defined node numbers must be unique within the connected controller.
SCAN_FTR		1	Without function when BLOCKED=1
SORT_REF		YES	REF1 - 24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID		1	1 after first successful transmission. Reset by user.
ERR			1 if the transmission has failed due to lost contact with destination node or queue full to CI535
REF 1	Value 1		DAT element to be sent to the RTU.
REF2	Value 2		DAT element to be sent to the RTU.
REF3	Value 3		DAT element to be sent to the RTU
REFn	Value n		DAT element to be sent to the RTU

Data Transmission from an RTU

MS for reception of data from RTUs are defined in the same way as normal receiving Data Sets (DS), except USER=3.

NOTE

Every receiving Data MS must have a corresponding sending Data MS to make the DATs accessible for the communication handler on the CI535 submodule. These MS must have IDENT = (IDENT (of receiving MS) - 100). In order to update only part of a Data MS, there must always be a sending Data MS (IDENT = 1-96) corresponding to a receiving Data MS (IDENT = 101-196). The sending and receiving Data MS are connected to the same DAT elements. When the CI535 submodule receives data from the RTU, the sending Data MS is requested from AC 400. The received data from the RTU is mapped into the data from the sending Data MS before it is sent back to the receiving Data MS in the AC 400.

Master Mode

The data is requested from the RTU with a Read Command MS (IDENT = 201) together with the CI535 submodule.

The transmission is controlled by a PC program (see [Section 3.6.7.4, Type Circuits for MVI Protocol Data Flow Control](#)) which sends a Read Command MS to the CI535 submodule. The CI535 submodule requests the sending Data MS containing the addressed registers and transmits the corresponding Read Command to the addressed RTU.

The reply from the RTU is mapped into the previously requested Data MS and returned to the corresponding receiving Data MS.

Slave Mode

A Write Command (function code 2) is received from the master containing data. The CI535 submodule requests the sending MS containing the addressed registers. The data from the master is mapped into the previously requested MS and returned to the corresponding receiving MS.

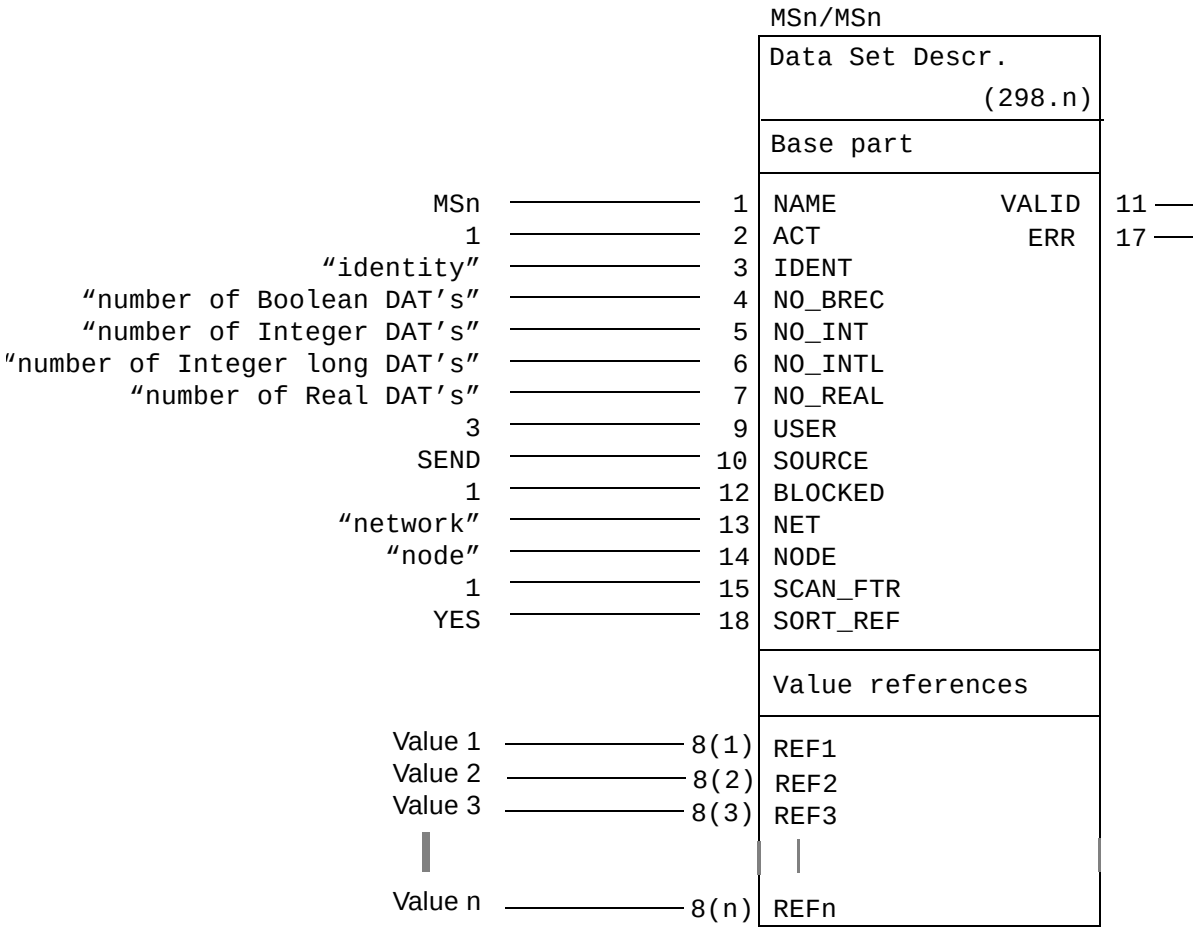


Figure 3-31. MVI Data Set for Reception of Data

The parameters REF1 - REF24 are stored in DAT elements referred to by this MVI Data Set. You can give the DAT elements arbitrary names.

The parameters have the following function:

Table 3-18. Description of MS for Reception of Data

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT			The identity of the MVI Data Set (101-196)
NO_BREC		0 - 24	Note: Only one type of REF could be defined.
NO_INT		0 - 24	Note: Only one type of REF could be defined.
NO_INTL		0 - 24	Note: Only one type of REF could be defined.
NO_REAL		0 - 24	Note: Only one type of REF could be defined.
USER		3	
SOURCE		RECEIVE	
BLOCKED		0	
NET		1 - 9	Network number of the MVI protocol bus. Defined in the CI535 data base element for the CI535 submodule. See Section 2.2.1, Hardware .
NODE		1 - 99	RTU number (node) of the receiving RTU. Valid node numbers must be defined in the Network Configuration MS. Note: All the defined node numbers must be unique within the connected controller.
SCAN_FTR		1	Used for reset of the VALID flag. VALID is set to 0 when the Time = 3*SCAN_FTR * MS_SCANTIME has passed. Default value for MS_SCANTIME = 2 s.
SORT_REF		YES	REF1 - 24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID		1	1 when the MS is updated. See also SCAN_FTR above.
ERR			1 if wrong data size is received
REF 1	Value1		DAT element from the RTU.
REF2	Value 2		DAT element from the RTU.
REFn	Value n		DAT element from the RTU.

Transfer of Appended Data MS

The MVI protocol functions (Command MS with IDENT=201-210 and 221-230) support data transfer from more than one Data MS in the same MVI protocol message, i.e., handling of more than 24/768 registers. If these MVI protocol functions must request or transmit data from two or more Data MS, the following must be accomplished:

- The MVI protocol register addresses must be consecutive in all Data MS used for one command (no holes are permitted). The last MVI protocol register address for the first Data MS must be followed directly by the start register address for the second Data MS, the last MVI protocol register address for the second Data MS must be followed directly by the start register address for the third Data MS, etc.
- All Data MS, except the last Data MS, used for one command must be built with 24 DATs.

PC Program Layout

To control the data flow on the asynchronous communication link, build a PC program in the AC 400. The PC program receives status information from the MVI protocol. The flow control is maintained through a chain of SENDREQ PC elements, each corresponding to a Command MS to be transmitted (see [Figure 3-33](#) and [Figure 3-34](#)).

Include all SENDREQ PC elements with Command MS addressed to the same RTU in the same chain to ensure a correct flow control to the MVI protocol link. Use the PC program to control the Command MS if the MVI protocol is used in master mode. A PC program is not necessary in slave mode.

The cycle time of CONTRM should be approximately 100 ms.

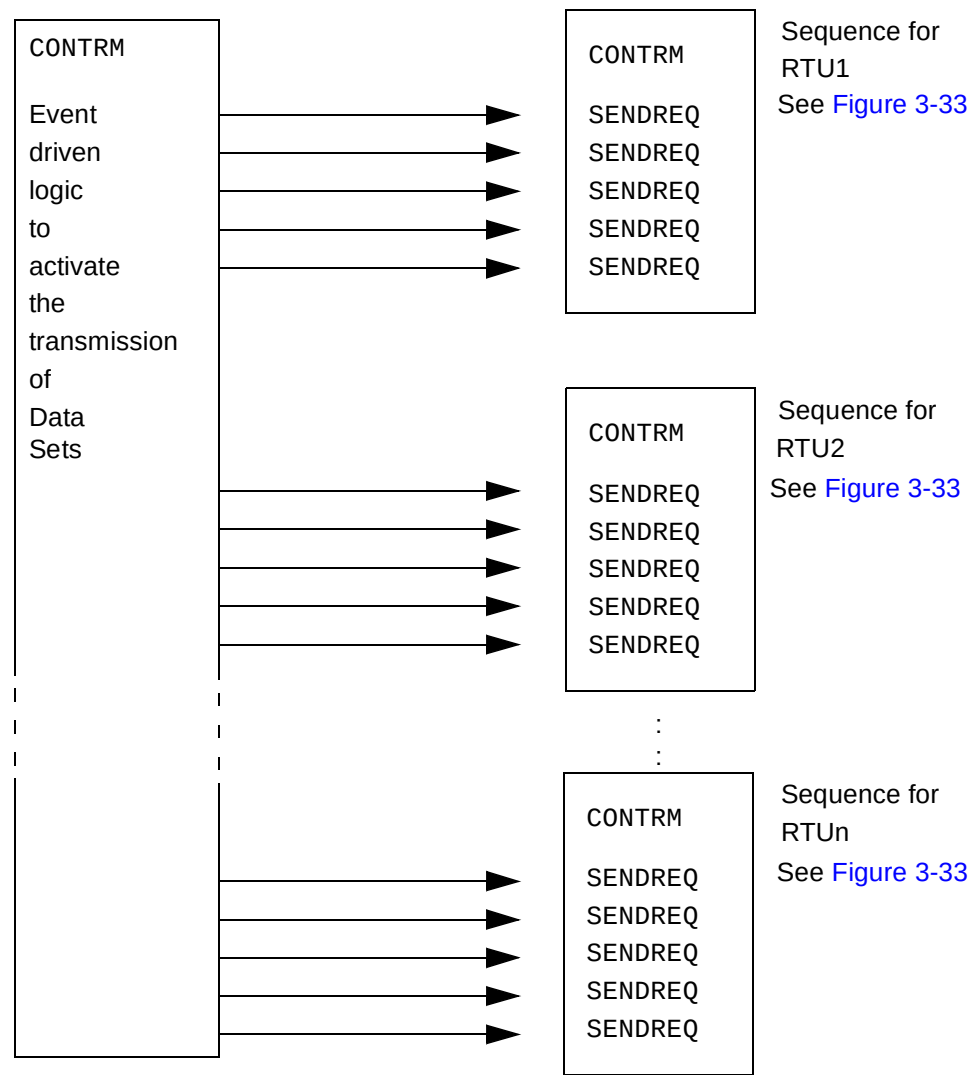


Figure 3-32. Overall PC Program Layout for an MVI Protocol in Master Mode

3.6.7.4 Type Circuits for MVI Protocol Data Flow Control

[Figure 3-33](#) shows a PC program type circuit which is used for data flow control on a MVI-MODBUS link. By linking several SENDREQ PC elements to each other with NXTBLK and PRVBLK, the order in which the data base elements are activated can be controlled. By using Link Status and Ready for Message status bits from the Status MVI Data Set to control the BLOCK terminal on the SENDREQ element, it can be assured that the CI535 submodule does not receive more messages than it can handle.

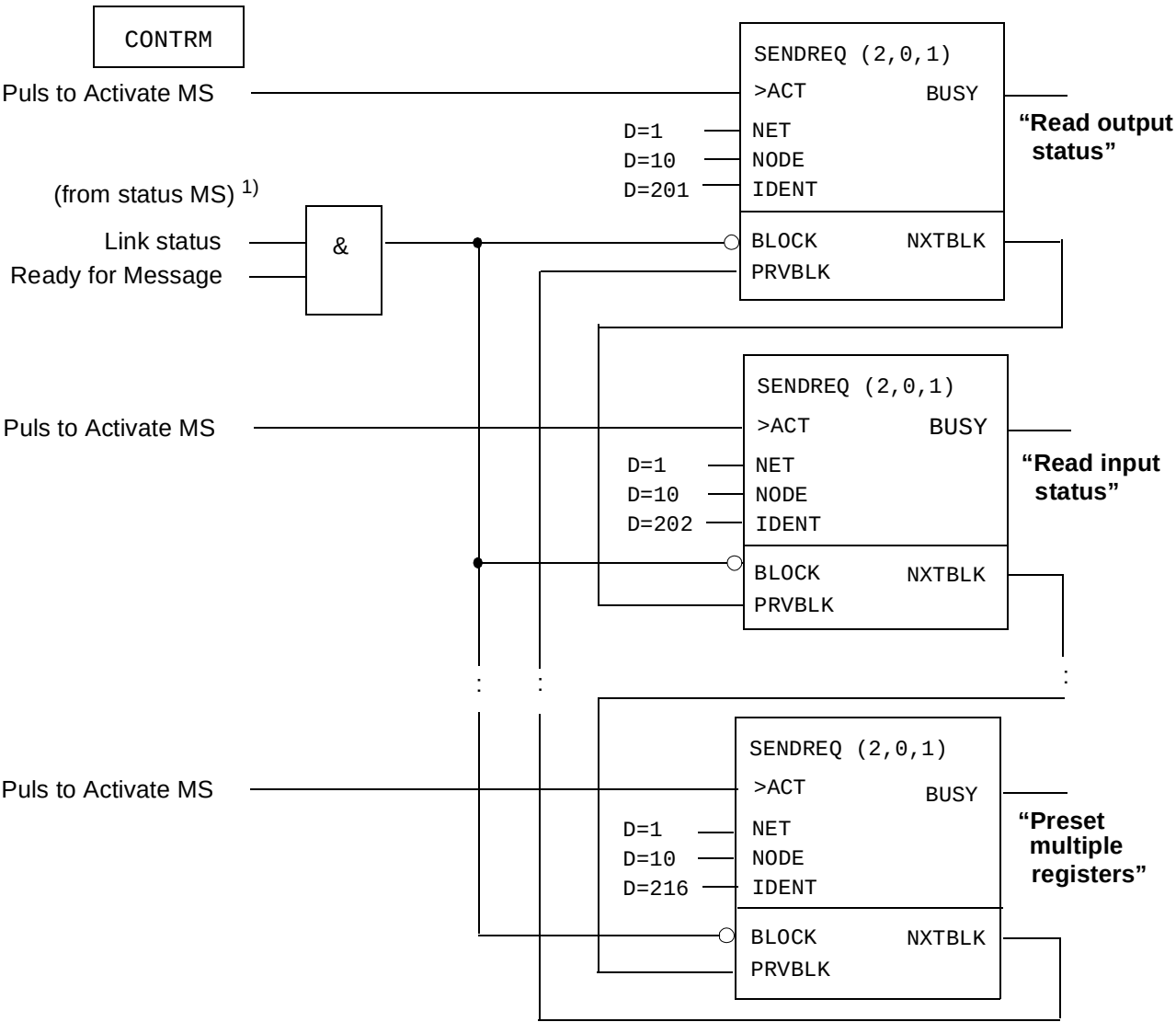
When ACT goes from 0 to 1, the MS with the identity given by NET, NODE and IDENT is activated once.

NXTBLK is true when sending of an MS is in progress. When sending is completed, it is set to false. When no sending is in progress, NXTBLK follows the value of PRVBLK.

PRVBLK is used together with NXTBLK to control the activation of the MS. No sending begins when PRVBLK is true. If sending orders are given at the ACT terminal during blocking by PRVBLK, the MS will be activated when PRVBLK is set to false.

No sending takes place when BLOCK is true. If sending orders are given at the ACT terminal during blocking by BLOCK, they are ignored.

Flow control of activation of MS is achieved by connecting the NXTBLK output of the first SENDREQ element to the PRVBLK input of next SENDREQ element. The NXTBLK output of this element is then connected to the PRVBLK input of the third SENDREQ element and so on. The NXTBLK output of the last SENDREQ element is connected to the PRVBLK input of the first SENDREQ element. Thus, the SENDREQ elements are connected in a chain, see [Figure 3-33](#). With this PC program sequence, each SENDREQ element will activate its MS, one for each execution cycle, in the order they are linked together.



1) Status is fetched from the Status MVI Data Set with IDENT = 3/13
for the PLC (in this example node=10), see [Section , MS for MVI Protocol Configuration](#)
Ready for Message = Status1:VALUE6
Link status = Status1:VALUE

Figure 3-33. PC Program Sequence for Control of an MVI Protocol in Master Mode

You can use this function block to multiplex different Register Addresses and number of registers into the Command MS DATs.

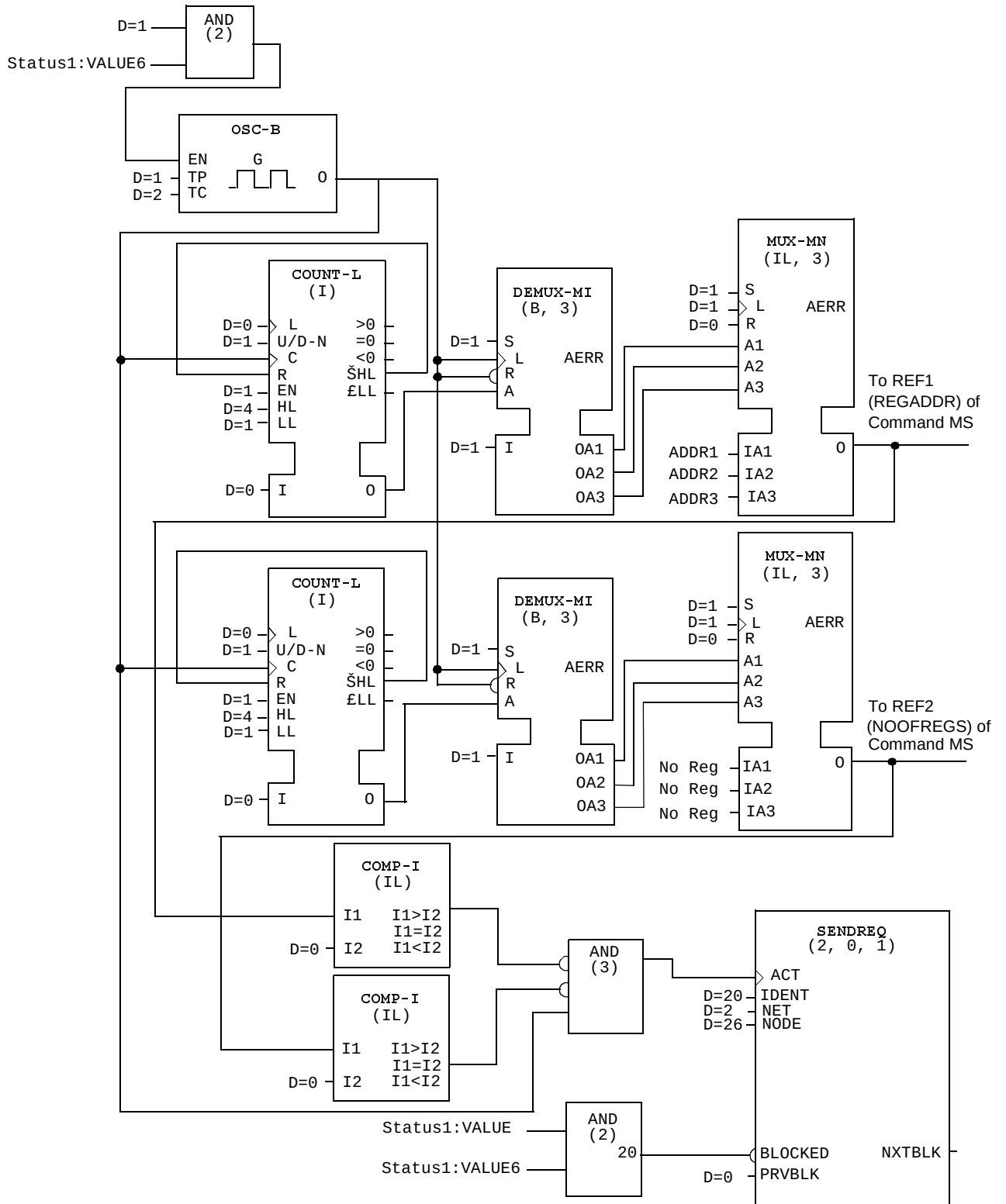


Figure 3-34. Function Block for Command MS

3.6.8 Host System Interface in Advant Controller 400

The Host System Interface provides functionality to transfer network-oriented data between the host system and an application on a communication submodule. The data could be locally addressed to a function on the submodule or remotely addressed to an application in another computer on the communication link. This data transfer is connection oriented, with one connection for each network-defined address on the communication submodule. A transceiver channel device is used for each connection. A buffer system is provided to achieve efficient data transfer.

NOTE

The Host System Interface is only needed for implementation of a Host Interface, for example, if the Advant Controller 400 Standard Stub cannot be used.

3.6.8.1 Network Addresses

A network address is defined when a transceiver channel device is created on the submodule. The address can be either remote or local.

Remote Address

A remote address is used for communication between an application in the host system and an application in another computer on the communication link. See [Figure 3-35](#).

```
typedef struct NWADDR
{
1)  u_int16  prio_gChIndx;      /* 2 .. 5          */
2)  int8     netw;             /* Network number  */
3)  int8     node;            /* Node number     */
} NetWAddr;
```

Figure 3-35. Remote Network Address

The following is a description of the fields in the NetWaddr struct for a remote address:

1. Priority on the network: 2 = Low priority, 3 = Normal priority, 4 = High priority and 5 = Emergency priority.
2. Destination network number. This should be a number between 1 and 9.
3. Destination node number. This should be a number between 1 and 99.

Local Address

A local address is used for communication between an application in the host system and a local function on the submodule. See [Figure 3-36](#).

```
typedef struct NWADDR
{
1)  u_int16  prio_gChIndx;    /* Global Channel Index */
2)  int8     netw;           /* Always = 0           */
3)  int8     node;           /* Always = -3          */
} NetWAddr;
```

Figure 3-36. Local Network Address

The following is a description of the fields in the NetWaddr struct for a local address:

1. Global destination channel index.
2. Always 0.
3. Always -3.

3.6.8.2 BufAllocate

NAME

bufAllocate - allocate a transceiver buffer

SYNOPSIS

STATUS bufAllocate(u_int16 size, BufPointer* buffer)

size, the size of the transceiver buffer. This is the actual size of a buffer allocated¹⁾.

buffer, where the buffer pointer is returned.

The files to be included are:

- vnsg01.h

DESCRIPTION

This routine allocates a transceiver buffer from the buffer pool. This is done by the write side before writing the transceiver buffer.

RETURNS

OK or ERROR if the call fails. If ERROR is returned, no buffer is allocated and errno indicates the cause of the error.

1. This size must be less than or equal to the nBytes argument of the transceiverInit() function.

3.6.8.3 BufRelease

NAME

bufRelease - release a transceiver buffer

SYNOPSIS

STATUS bufRelease(struct Buf* BufPointer buffer)

buffer, the pointer to the buffer to release.

The files to be included are:

- vnsg01.h

DESCRIPTION

This function releases a transceiver buffer to the buffer pool. This is done by the read side when the transceiver buffer is consumed.

RETURNS

OK or ERROR if the call fails. If ERROR is returned, the buffer is not released and errno indicates the cause of the error.

3.6.8.4 TransceiverInit

NAME

transceiverInit - init function

SYNOPSIS

STATUS transceiverInit(u_int16 maxConNumber, u_int16 nBytes, u_int16 nBuf)

maxConNumber, maximum number of connections over the Host System Interface

nBytes, maximum size of a transceiver buffer¹⁾

nBuf, number of transceiver buffers in the buffer pool

The files to be included are:

- vnsg01.h

DESCRIPTION

This function initiates the Host System Interface and its data structures. This function must be called before the Host System Interface is used.

RETURNS

OK or ERROR if the call fails. If ERROR is returned, the buffer is not released and errno indicates the cause of the error.

1. This is the absolute maximum size of a transceiver buffer. Note that you can allocate a buffer which is smaller than the maximum size using bufAllocate. See [Section 3.6.8.2, BufAllocate](#).

3.6.8.5 TranHostInfoGet

NAME

tranHostInfoGet - fetch host information

SYNOPSIS

tranHostInfoGet(HostInfo* hostInfoP)

hostInfoP, where the host information is returned

The files to be included are:

- vnsg01.h

DESCRIPTION

This function returns the host information, that is, the submodule network address (submodule number), channel 1 network address and channel 2 network address. See data structures in vnsg01.h for details.

RETURNS

OK or ERROR if the call fails. If ERROR is returned, errno indicates the cause of the error.

3.6.8.6 TranChanDevCreate

NAME

tranChanDevCreate - create a transceiver channel device

SYNOPSIS

tranChanDevCreate(char* name, u_int16 netwType, NetWAddr* netwAddrPtr,
u_int16 nMessage, u_int16 nBytes)

name, device name used by open system call

netwType, network type: 7 = RCOM, 8 = all other MVI protocols

netwAddrPtr, network address for the remote unit, see [Section 3.6.8.1, Network Addresses](#) for details

nMessage, number of positions in the buffer queue

nBytes, size of the messages which are to be transferred through the transceiver device. This must be less than or equal to the transceiver buffer size given when calling transceiver Init.

The files to be included are:

- vnsg01.h

DESCRIPTION

There are transceiver channel devices associated with the Host System Interface. These are the devices which are used for the communication. One device instance is to be created for each connection. The transceiver channel devices fulfill the UNIX compatible I/O interface, that is, open read write ioctl select and close.

RETURNS

OK or ERROR if the call fails. If ERROR is returned, errno indicates the cause of the error.

3.6.8.7 Open

NAME

open - open a transceiver channel device

SYNOPSIS

```
int open(char* name, int flag, int mode)
```

name, name of the device to be opened

flag, a flag field describing the mode in which the file is opened. The value is constructed by OR-ing flags described below:

O_RDONLY open for reading only

O_WRONLY open for writing only

O_RDWR open for reading and writing

Mode not used.

The files to be included are:

- vnsg01.h

DESCRIPTION

This routine opens a transceiver channel device. The device to be opened must be created using tranChanDevCreate.

RETURNS

The file descriptor, which is a global identifier for the channel device, or ERROR if the call fails. If ERROR is returned, errno indicates the cause of the error.

3.6.8.8 Read

NAME

read - read a transceiver buffer pointer from a transceiver channel device

SYNOPSIS

```
STATUS read(int fd, BufPointer* bufPtrP, u_int16 length)
```

fd, file descriptor for the device from which to read

bufPtrP, pointer to where to put pointer read

length, size of the data to be read

The files to be included are:

- vnsg01.h

DESCRIPTION

This routine reads a transceiver buffer pointer from a transceiver channel device. After using the data in the buffer received from the host system, the buffer must be released using bufRelease().

RETURNS

The length of data read, which is four due to the fact that there is always a pointer read, or ERROR if the call fails. If ERROR is returned, errno indicates the cause of the error.

3.6.8.9 Write

NAME

write - write a transceiver buffer pointer to a transceiver channel device

SYNOPSIS

```
STATUS write(int fd, BufPointer* bufPtrP, u_int16 length)
```

fd, file descriptor for the device from which to read

bufPtrP, pointer to the transceiver buffer pointer that is to be written

length, size of the data to be written

The files to be included are:

- vnsg01.h

DESCRIPTION

This routine writes a transceiver buffer pointer to a transceiver channel device. Before writing the transceiver buffer pointer to the host system, the transceiver buffer must be allocated using bufAllocate() and filled in with data.

RETURNS

The length of data read, which is four due to the fact that there is always a pointer read, or ERROR if the call fails. If ERROR is returned, errno indicates the cause of the error.

3.6.8.10 Close

NAME

close - close a transceiver channel device

SYNOPSIS

```
void close(int fd)
```

fd, file descriptor for the device to be closed

The files to be included are:

- vnsg01.h

DESCRIPTION

This routine closes a transceiver channel device.

RETURNS

OK or ERROR if the call fails. If ERROR is returned, errno indicates the cause of the error.

Chapter 4 Runtime Operation

4.1 Operating Overview

Advant Controller 400 must be in operation mode (mode P1, indicated on the front of the main processor module) to communicate on the MVI link. With Advant Controller 400 in configuration mode (mode P2, indicated on the front of the main processor module), the communication is stopped.

After start-up, and with Advant Controller 400 in operation mode, you can perform the following operations:

- From the PC program for activation of commands and flow control
 - Activate/Stop sending of Command MS (Read, Write and Status Request command).
 - Re-initiate a channel on the CI535 submodule by sending a restart handler command.
 - Restart the CI535 submodule by setting/re-setting the terminal SERVICE on the CI535 data base element.
- The status information for a node on the MVI link in corresponding RTU Status MS can be displayed.

4.2 Runtime Tutorial

A Runtime Tutorial is not included in this manual. The user must be familiar with application commands in Advant Controller 400.

4.3 Operating Instructions

For a description of commands for control of the PC program and data base, please refer to *AMPL Configuration Advant Controller 400 Series Reference Manual*.

Chapter 5 Maintenance

5.1 Preventative Maintenance

Check the stack usage for the tasks comprising the protocol implemented during runtime. To do this, use the command `checkStack` from the console.

For example:

```
-> checkStack tMviHdl01
NAME          ENTRY          TID          SIZE    CUR    HIGH    MARGIN
-----
tMviHdl01     _mviXYMain  23e1c78    4000    832    3968    32
```

where the different entities are:

- NAME - task name.
- ENTRY- task entry point, i.e., the main function.
- TID - task identity.
- SIZE - the total stack size.
- CUR - the current number of stack bytes used.
- HIGH - the maximum number of stack bytes used.
- MARGIN - the number of bytes never used at the top of the stack (SIZE - HIGH).

The margin should not be less than 25 percent of the total stack size during normal operation.

5.2 Hardware Indicators

The CI535 submodule has two LED indicators on the front:

- The green LED is lit when the data base element CI535 for the CI535 submodule is correctly filled in, see [Section 2.2.1, Hardware](#) (the terminal IMPL and SERVICE must be set to 1 on the CI535 element).
- The red LED is lit when a fatal error is detected on the CI535 submodule.

See [Figure 5-1](#) below.

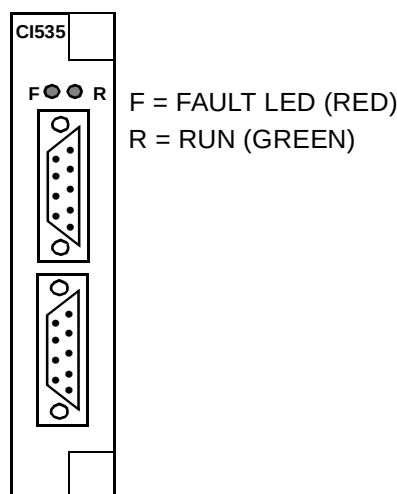


Figure 5-1. LED Indicators on CI535 Submodule

5.3 Error Messages

This part of the document lists all **non-protocol specific** system messages that might be generated from the CI535 submodule. System messages from the Advant Controller 400 system software are not included. The MVI Data Set task CXAM000 can report a number of errors related to the MVI communication – MS not found, destination unreachable, channel to handler full and so on. For translation of system messages from the Advant Controller 400, please refer to the *Advant Controller 410 User's Guide* or *Advant Controller 450 User's Guide*.

The error codes indicated on the data base element CI535 are translated in [Table 5-1](#) below.

Table 5-1. Error Codes from the Data Base Element CI535

Error Code	Interpretation
21	Duplicate node number on the network
22	More than one node number in the station
23	Station connected to more than one Control Network
25	Network number already used in the Advant Controller
31	Module not found or internal bus error
32	Incorrect hardware identity on module
33	Start up of module failed
34	Suspected hardware error

Table 5-1. Error Codes from the Data Base Element CI535 (Continued)

Error Code	Interpretation
35	Incorrect software identity on module
36	Incorrect module interface identity
50	Illegal slave number
51	Illegal module position
52	Module position number already used in the Advant Controller
53	Two modules have the same slave number

System message printouts on the engineering station have the following appearance:

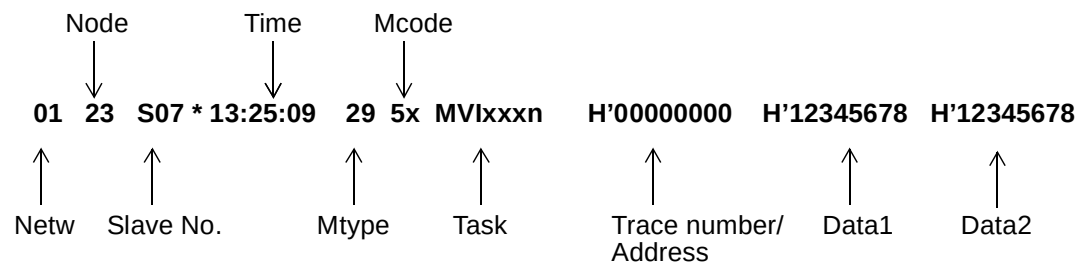


Figure 5-2. System Message Layout

All system messages have:

- Mtype = 29
- Task name = MVIxxxxn
 - xxx = HDL if the MVI Protocol Handler reports the error
 - xxx = DLL if the MVI Protocol Handler Data Link layer reports the error
 - xxx = NWL if the MVI Protocol Handler Network layer reports the error
 - xxx= STB if the MVI Protocol Stub reports the error
 - xxx = MUX if the MVI Protocol Multiplexer reports the error
 - n = 1 for port one
 - n = 2 for port two

Table 5-2. Messages with Mcode = 55 and Task Name = MVIHDLn from CI535

Trace Number (Hexadecimal)	Parameter Description 1 = data1 2 = data2	Comments and Suggested Actions
00010002	1: - 2: -	Failed to create UART device. Reset the MVI module, if the problem persists, please contact ABB.
00010003	1: File descriptor 2: -	Initiate UART failed. Check that the UART device is properly created and that the line characteristic parameters are set up correctly.
00010004	1: - 2: -	Failed to open UART device. Check that the UART device is properly created.
00010009	1: File descriptor 2: -	Parameterize UART failed. Check that the line characteristic parameters are set up correctly.
00020002	1: - 2: -	Failed to create Watchdog Timer device. Reset the MVI module, if the problem persists, please contact ABB.
00020004	1: - 2: -	Failed to open Watchdog Timer device. Check that the Watchdog Timer device is properly created.
00040002	1: - 2: -	Failed to create FIFO device. Reset the MVI module, if the problem persists, please contact ABB.
00040004	1: - 2: -	Failed to open FIFO device. Check that the FIFO device is properly created.
00040005	1: File descriptor 2: -	Read from FIFO failed. Check that the FIFO device is properly created and opened.
00040006	1: File descriptor 2: -	Write to FIFO failed. Check that the FIFO device is properly created and opened.
0005000f	1: Requested block size 2: -	Failed to allocate memory. Check that the malloc or calloc call is properly coded.
00050012	1: - 2: -	Select failed. Check that the select call is properly coded and that the devices set up in the select masks are properly created and opened.
00060013	1: Node number 2: -	Illegal node number. The node number requested was not configured. Mismatch between network configuration and command MS. Check the configuration.

Table 5-3. Messages with Mcode = 56 and Task Name = MVISTBn from CI535

Trace Number (Hexadecimal)	Parameter Description 1 = data1 2 = data2	Comments and Suggested Actions
00020002	1: - 2: -	Failed to create Watchdog Timer device. Reset the MVI module, if the problem persists, please contact ABB.
00020004	1: - 2: -	Failed to open Watchdog Timer device. Check that the Watchdog Timer device is properly created.
00030004	1: File Descriptor 2: -	Failed to open transceiver channel. Check that the transceiver channel device is properly created.
00030005	1: File Descriptor 2: -	Read from transceiver channel failed. Check that the transceiver channel device is properly created and opened.
00030006	1: File Descriptor 2: -	Write to transceiver channel failed. Check that the transceiver channel device is properly created and opened.
00030007	1: File Descriptor 2: -	Failed to close transceiver channel. Reset the MVI module, if the problem persists, please contact ABB.
0003000f	1: File Descriptor 2: -	Failed to allocate transceiver buffer. Check the initialization of the Host System Interface with respect to number of transceiver buffers in the buffer pool. Check also that the transceiver buffers read are released after use.
00030016	1: - 2: -	Failed to fetch Host Information. Check that the CI535 data base element is properly filled in and that the Advant Controller is in operation mode.
00040004	1:- 2: -	Failed to open FIFO device. Check that the FIFO device is properly created.
00040005	1: File Descriptor 2: -	Read from FIFO failed. Check that the FIFO device is properly created and opened.
00040006	1: File Descriptor 2: -	Write to FIFO failed. Check that the FIFO device is properly created and opened.
0005000f	1: Requested block size 2: -	Failed to allocate memory. Check that the malloc or calloc call is properly coded.
00050012	1: - 2: -	Select failed. Check that the select call is properly coded and that the devices set up in the select masks are properly created and opened.

Table 5-3. Messages with Mcode = 56 and Task Name = MVISTBn from CI535 (Continued)

Trace Number (Hexadecimal)	Parameter Description 1 = data1 2 = data2	Comments and Suggested Actions
00070014	1: Data1 = xxyyyyyy xx = 1: Duplex (REF7) xx = 2: Speed (REF3) xx = 3: Character Length (REF4) xx = 4: Stop bit (REF5) xx = 5: Parity (REF6) xx = 6: Illegal RTU type (REF1) xx = 7: Illegal Modem type (REF17) xx = 8: Illegal MS identity yyyyyy = parameter value 2: -	Illegal Line Characteristics parameter. Check the corresponding DAT value in the Line Characteristics MS. Restart the CI532V02 submodule after change of Line Characteristics MS.
00070018	1: MS identity 2: -	Time-out waiting for Configuration MS. Create the Configuration MS with ident=Data1, and fill in the connected DAT references. See Section 3.6.8.1, Network Addresses .
00070019	1: RTU register address 2: Number of registers requested	Illegal data request. The Register Address MS must contain the register address = Data1. Fill in the correct register address and restart the CI532V02 submodule. The Data MS corresponding to the RTU register address must also be configured in the AC 400. If the RTU register address is not to be used, then check the DAT values for the Command MS or the command from the RTU.

Table 5-4. Messages with Mcode = 57 and Task Name = MVIMUXn from CI535

Trace Number (Hexadecimal)	Parameter Description 1 = data1 2 = data2	Comments and Suggested Actions
00030002	1: - 2: -	Failed to create transceiver channel. Reset the MVI module, if the problem persists, please contact ABB.
00040004	1: - 2: -	Failed to open FIFO device. Check that the FIFO device is properly created.
00040005	1: File Descriptor 2: -	Read from FIFO failed. Check that the FIFO device is properly created and opened.
00040006	1: File Descriptor 2: -	Write to FIFO failed. Check that the FIFO device is properly created and opened.
00070015	1: m_code (ident) 2: -	Illegal DSX Signal received. Check the m_code (ident) for all configuration MS and command MS.

Table 5-5. Messages with Mcode = 58 and Task Name = MVIHDLn from CI535

Trace Number (Hexadecimal)	Parameter Description 1 = data1 2 = data2	Comments and Suggested Actions
00010005	1: File Descriptor 2: -	Read from UART failed. Check that the UART device is properly created and opened and that the Read Descriptor is properly filled in.
00010006	1: File Descriptor 2: -	Write to UART failed. Check that the UART device is properly created and opened and that the write descriptor is properly filled in.
00020004	1: - 2: -	Failed to open Watchdog Timer device. Check that the Watchdog Timer device is properly created.
00050012	1: - 2: -	Select failed. Check that the select call is properly coded and that the devices set up in the select masks are properly created and opened.

5.4 Fault Finding and User Repair

5.4.1 SLIP Communication

If you have followed the instructions in [Chapter 2, Installation](#) and [Chapter 3, Configuration/Application Building](#), and the link between host and target machines still does not work, please check the following:

1. It is not possible to “rlogin” to, or “ping” the CI535 Target.

Check that PPL is correctly configured. This is done in the `/usr/lib/ppl/ppl.remotes` file. See [Section 3.3.4, SLIP Interface](#) for details.

Check that the Internet addresses for host and target are correctly configured in both host and target. See [Section 3.3.3, Host/Target Internet Addresses](#) for details.

Check that the PPL network interface device file `/dev/ni` exists and is correctly defined using the command:

\$ ls -l /dev/ni

the major number should be 56 and the minor number 0. See [Section 3.3.4, SLIP Interface](#) for details.

Check that the serial port device file used for SLIP connection, `/dev/ttyxx`, exists and is correctly defined using the command:

\$ ls -l /dev/tty*

the major and minor number values depend on a number of details concerning your HP9000/700 work station. See your HP9000/700 system administrator for details.

Check that ppl is running using the command:

\$ ps -elf | grep “ppl -o slip_target” This should result in something like:

```
1 S lla 27268 3590 3 154 20 baef380 15 453994 16:19:51 ttyq1 0:00 ppl -o <
slip_target.1)
```

Check that the MVIBase system on the CI535 submodule is a development system, that is, a system with support for networking, SLIP and remote debugging. See [Section 3.5.1.5, Build a Prommed System](#) for details.

Check that the SLIP communication cable between host and target is correctly wired. See [Section 2.2.1, Hardware](#) for details.

2. You can “rlogin” to the Target, but you cannot, from the Target, access the File System on the Host.

The CI535 target name (slip_target) must be entered in either `/etc/hosts.equiv` or `$HOME2)/.rhosts` on your HP9000/700 work station.

In `/usr/adm/inetd.sec`, you must define that the CI535/CI536 target is allowed to use the remote services shell (remote shell), login (remote login) and rexd (Remote Procedure Calls) to access the HP9000/700 host.

-
1. For details on how to interpret the output from the ps command, refer to the on-line manual page for ps (man ps). **Note** that this output line is just an example. The values for the different fields will vary from time to time.
 2. Assuming that korn shell is used, \$HOME is equivalent to the developer's login directory.

5.4.2 Cross-Development Tools

5.4.2.1 Editor

If your favorite editor is not invoked by the proper command, the path may not be defined in **\$PATH**, or it may not be installed on your HP9000/700 work station. Talk to your HP9000/700 system administrator for assistance.

5.4.2.2 Compiler

If the compiler is not invoked by the proper command or working correctly, check the following:

- Is MVIBase properly installed? Installation of MVIBase is described in [Section 2.2.2.1, MVIBase and User Development Structure](#).
- Is the path to cross-development tools `/ipa/products/MVIBase/2.1-0/bin` set up in the **PATH** environment variable?
- Is the environment variable **GCC_EXEC_PREFIX** defined to `/ipa/products/MVIBase/2.1-0/bin/`?
- Is the environment variable **VX_CPU_FAMILY** defined to `68K`?

5.4.2.3 Debugger

If VxGDB is not invoked by the proper command or working correctly, check the following:

- Is MVIBase properly installed? Installation of MVIBase is described in [Section 2.2.2.1, MVIBase and User Development Structure](#).
- Is the path to cross-development tools `/ipa/products/MVIBase/2.1-0/bin` set up in the **PATH** environment variable?
- Is the environment variable **GCC_EXEC_PREFIX** defined to `/ipa/products/MVIBase/2.1-0/bin/`?
- Is the softlink named `vxWorks.st` in the developer's login directory created and pointing to the load module generated? See [Section 3.3.7, Debugger Initialization](#) for details.

5.4.3 MVI Demo Protocol and other Asyn. Communication Protocols

If you followed the instructions in [Chapter 2, Installation](#) and [Chapter 3, Configuration/Application Building](#) and still do not have a correctly working communication link, please check the following:

1. Are Modems and Cables correctly Set-up?

The CI535 must receive the correct modem signals. When you select full duplex mode for the modem signals, the signals Data Set Ready (DSR), Data Carrier Detect (DCD) and Clear to Send (CTS) must be set. In half duplex mode, the DSR must be set and DCD and CTS must toggle between 0 and 1.

Current DSR, DCD and CTS values are available in the data base element CI535.

NOTE

Displayed values from the CI535 element on the engineering station are not synchronized to the actual value change of the signals; short pulses may not be displayed.

The modem signals are set up by the modem or, in full duplex mode, you can wire them in the 25-position connector on cable TK577 (see [Figure 2-5](#)).

2. Is the CI532 Data Base Element correctly Filled in?

You must define the same node number for all the communication boards in an Advant Controller 400 node. All network numbers in the Advant Controller 400 node must be unique (for CI535 1-9). If the network number or node number for any of the communication modules in the Advant Controller 400 is changed after the start-up of the node, the Advant Controller 400 must be COLD STARTED. (Connect an engineering station to the Advant Controller 400 and make a DUAP, press ENTER on the main CPU with start-up switch in 'CLEAR' position and make a LOAP.)

3. Check the Data Base and PC Program Application.

- a. Have the Configuration MS (Line Characteristics, Network Configuration, Register Address MS and RTU Status MS) been sent to the CI535 submodule? The VALID flag on these MS is set to 1 when the MS is successfully transferred to the module.

The Record number on the CI535 data base element defines the submodule number used for the NET terminal on the Configuration MS. The submodule numbers are 7 to 11 in Advant Controller 450 and 7 and 8 in Advant Controller 410.
- b. Data MS for the actual data transfer must be defined. In order to update only a part of a MS, there must always be a sending MS (IDENT = 1-96) corresponding to a receiving MS (IDENT = 101-196). The sending and receiving Data MS must be connected to the same DAT elements.
- c. In master mode, the Command MS and the PC program with SENDREQ-elements for the flow control must be built and started.

4. Are there any System Messages reported in the Advant Controller 400?

Check system messages by connecting an engineering station to the Advant Controller 400. Also check the system messages when you restart the CI535. To start the module, set the terminal SERVICE to 0 and back 1 in the CI535 element. All configuration errors are reported at start-up. Error messages with task name MVIxxx and mtype 29 are translated in [Section 5.3, Error Messages](#).

NOTE

Some errors are only reported once.

5. What Status Bits are set in the RTU Status MS?

VALUE (bit 0) is set when the Advant Controller 400's connection to the RTU is established, see [Section 3.6.7.1, MS for MVI Protocol Configuration](#).

If you know the answers to the following questions, further analysis will be easier:

- a. Are modems and cables installed as described in the *Advant OCS Installation Rules* manual? Pay particular attention to the power supply and connection to signal ground.
- b. How is your control system configured? A block diagram with all node and network numbers is useful. Note also the type of RTU, the master and slaves on the link and the type of modem you are using.
- c. Have you made any significant changes in your configuration (for example, new communication board(s) added, new modems)?
- d. Is it possible to repeat your problem?
- e. Have you found any way to circumvent the problem?
- f. Have you made any other observations related to the problem?

A test tool for monitoring the status of the V.24 leads is also useful for further analysis. The test tool shows the status, with LED indicators, of the modem signals, data transmission (TD) and reception (RD). When an analysis of the data flow on the communication link is necessary, you need a Data Link Analyzer that monitors and stores the signals sent on the link.

NOTE

Some of the points in the list above are not applicable on the MVI Demo Protocol alone, but may be relevant for any asynchronous communication protocol developed.

Appendix A Code Example

A.1 Example: Protocol Handler

An example of a Protocol Handler implemented as a state machine follows. This example is not a complete Protocol Handler, but demonstrates the important parts, that is, how different MVI Service Routines are called and in which order, how the MVI Protocol Handler Interface is used, handshaking between Protocol Handler and Protocol Stub, and so forth.

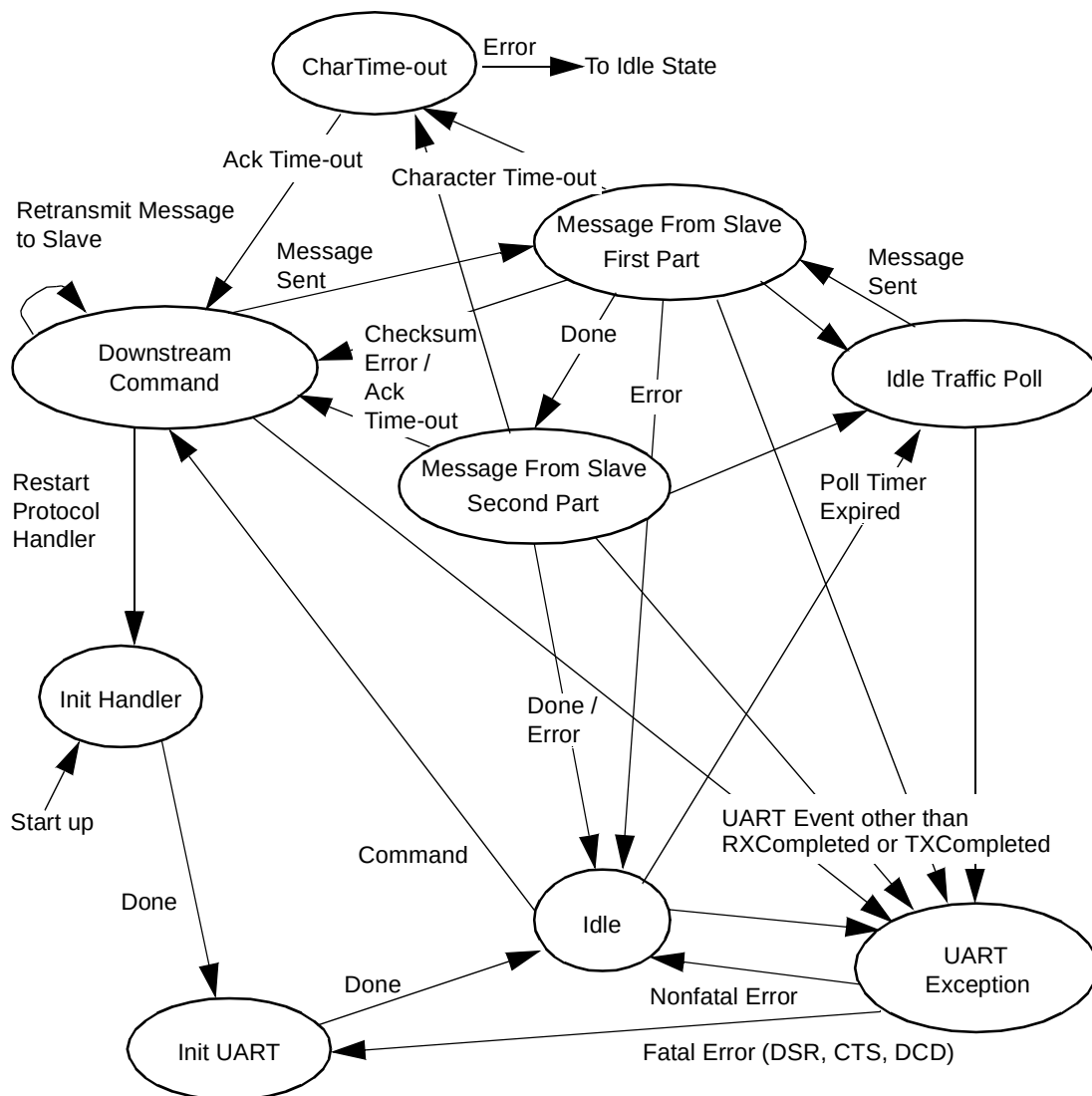


Figure A-1. Master Mode State Diagram

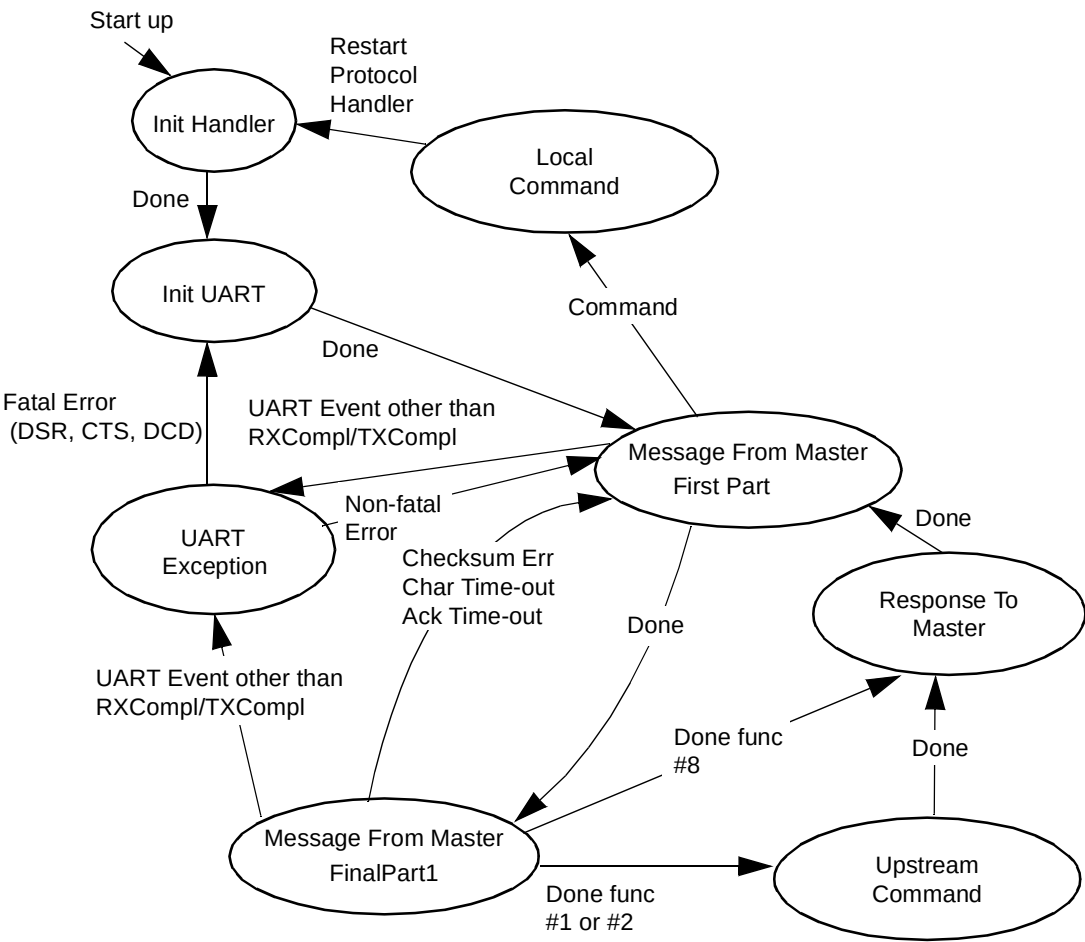


Figure A-2. Slave Mode State Diagram

NOTE

Although the state diagram for slave mode is shown here, the code example only shows the master mode. For details about slave mode, look at the source code for the Demo Protocol included in MVI.

mviXYMain()

```
void mviXYMain(int taskInstance)
{
    int            i;
    MviDL          dlStruct;
    XYGlobalData   globalData;

    while (1)
    {
        if (xyHandlerInit(&dlStruct, &globalData,
                           taskInstance) == MviError)
        {
            exit(1);
        }
        if (dlStruct.setupDataPtr->master == TRUE)
        {
            xYMasterLoop(&dlStruct, &globalData);
        }
        else
        {
            xYSlaveLoop(&dlStruct, &globalData);
        }
        /* Close Idle traffic poll timers
        (one for each node on the link) */

        for (i = 0; i < dlStruct.usedNodes; i++)
        {
            close(((XYDLEntry*) dlStruct.dlEntry)[i].pollTimerFd);
        }
    }
}
```

This function (mviCleanup) frees memory for data structures allocated and closes devices previously opened by MVI support routines (mviFetchSetupInfo, mviInitDLStruct, mviInitCommonGlobalDataStruct and mviOpenPHIDevices). The pointers in the MviDL struct CANNOT be used until a new initialization has been performed.

```
    mviCleanup(&dlStruct, (MviGlobalData*) &globalData,
               MviProtHandler);
}
```

xYHandlerInit()

```
MviStatus xYHandlerInit(MviDL* dlPtr,  
                        XYGlobalData* globalDataPtr,  
                        int taskInstance)  
{  
    /*-----  
    Open pipes to protocol stub  
    -----*/
```

This function (mviOpenPHIDevices) opens the Protocol Handler Interface FIFO devices, used by the Protocol Handler to communicate to the Host System. This function must be called before any communication with the Host System can be made.

```
    if(mviOpenPHIDevices((MviGlobalData*)globalDataPtr,  
                        taskInstance) != MviOk)  
    {  
        return(MviError);  
    }  
  
    /*-----  
    Fetch setup information from host system  
    -----*/
```

This function (mviFetchSetupInfo) fetches the initialization information pointer from the Host System Interface. The mviOpenPHIDevices function must be called before this function, since the Protocol Handler Interface is used, and must be initialized. The function must be called before mviInitDLStruct, since the setup information is needed for the DL structure data.

```
    if (mviFetchSetupInfo(dlPtr, (MviGlobalData*) globalDataPtr)  
        != MviOk)  
    {  
        return(MviError);  
    }  
  
    /*-----  
    Allocate data structures associated to the DL-structure and  
    initiate the DL-structure, status info structure and  
    diagnostic info structure  
    -----*/
```

This function (mviInitDLStruct) allocates data structures associated to the DL structures (MviInitDescriptor, MviUARTParBlk, MviReadDesc, MviDataBuffer(read and write), MviBuffer, MviWriteDesc, MviDataBuffer, MviDiagInfo, MviNodeDiag, MviStatInfo, and MviNodeStat) and initializes the DL structure, and the common part of each DL-entry. Setup data is used from mviFetchSetupInfo.

```
if (mviInitDLStruct(dlpPtr, 1, 2, sizeof(XYDLEntry),
                  taskInstance, MviProtHandler) != MviOk)
{
    return(MviError);
}

/*-----
Initiate standard global data
-----*/
```

This function (mviInitCommonGlobalDataStruct) allocates the send command buffer (MviCmndInfo) to 'sendCmndPtr', and initializes 'wakeupReason' in the standard part of the common global data structure.

```
if (mviInitCommonGlobalDataStruct(
    (MviGlobalData*)globalDataPtr) != MviOk)
{
    return(MviError);
}

globalDataPtr->state = InitUart;

/*-----
Open UART device for read and write
-----*/
```

This function (mviOpenUARTDevice) opens the UART device corresponding to a given task instance. The device is closed by the mviCleanup function.

```
if (mviOpenUARTDevice(&globalDataPtr->comData.fdTab[UartFd],
                    taskInstance) != MviOk)
{
    return(MviError);
}

/*-----
Open timer devices
-----*/
```

This function (mviOpenTimerDevice) opens the TXTimer, WatchdogTimer and AckTimer devices. The devices are closed by the mviCleanup function.

```
if (mviOpenTimerDevices(dlPtr, (MviGlobalData*)globalDataPtr,  
                        MviProtHandler)) != MviOk)  
{  
    return(MviError);  
}  
  
/*-----  
    Initiate protocol specific part of global data  
-----*/  
  
if ((initProtocolData(dlPtr, globalDataPtr)) != MviOk)  
{  
    return(MviError);  
}
```

This function (mviSendDiagInfo) sends the diagnostics info pointer to the Protocol Stub, which sends the diagnostics info to the Host System on demand.

```
if(mviSendDiagInfo(globalDataPtr->comData.fdTAb[DiagInfoFd],  
                  &(dlPtr->diagInfoPtr)) != MviOk)  
{  
    return(MviError);  
}  
  
return(MviOk);  
}
```

xYMasterLoop()

```
void xYMasterLoop(MviDL* dlPtr, XYGlobalData* globalDataPtr)
{
    int          i;
    int          width;
    MviFdSet     fdsetR;
    DxBoolean    selectedFdNotFound;
    DxBoolean    readDescIsCanceled;
    int          watchdogTime = 5000L;

    /*-----
       Start Idle Poll Timers using startIdlePollTimers
       -----*/
    if ( startIdleTrafficPollTimers(dlPtr, globalDataPtr)!=MviOk)
    {
        exit(2);
    }

    /*-----
       Start System Watchdog Timer
       -----*/

    if (ioctl(globalDataPtr->comData.fdTAb[WatchdogTimerFd],
              MviCWDTStart, (int) &watchdogTime) < 0)
    {
        /* System message */
        exit(3);
    }

    while (globalDataPtr->state != InitHandler)
    {
        switch (globalDataPtr->state)
        {
            case InitUart:
```

This function (mviInitUART) initializes the UART parameter block, the resume mask and the UART device according to setup data and input parameters.

```
        if (mviInitUART(((
            MviGlobalData*)globalDataPtr>fdTAb[UartFd],
            dlPtr->setupDataPtr,
            dlPtr->parameterBlockPtr,
            dlPtr->initDescriptorPtr,
            DxFalse, DxFalse, DxFalse, DxFalse) != MviOk)
        {
            exit(4);
        }
```

This function (`mviWaitForModemSignals`) waits for the modem signals DCD, CTS and DSR to become active and stable.

```
mviWaitForModemSignals(dlPtr,  
                        (MviGlobalData*)globalDataPtr);  
globalDataPtr->state = Idle;  
  
/* Fall through to next state */  
case Idle:
```

This function (`FD_ZERO`) clears the select mask, the address for which is given as argument.

```
FD_ZERO(&fdsetR);  
width = (globalDataPtr->comData.fdTAb[UartFd] >  
         globalDataPtr->comData.fdTAb[DownStreamCmndFd]) ?  
         globalDataPtr->comData.fdTAb[UartFd] :  
         globalDataPtr->comData.fdTAb[DownStreamCmndFd];
```

This function (`FD_SET`) sets a bit corresponding to the file descriptor given as the first argument, in the select mask for which the address is given as the second argument.

```
FD_SET(globalDataPtr->comData.fdTAb[UartFd], &fdsetR);  
FD_SET(globalDataPtr->comData.fdTAb[DownStreamCmndFd],  
        &fdsetR);
```

This function (`mviSelect`) handles the Watchdog message sending to the Protocol Stub and calls select to wait for an event from one or several of the devices for which file descriptor is set in the read and/or write select mask. When an event occurs, `mviSelect` returns with the file descriptor set in the read or write mask for the device or devices in question.

```
mviSelect(width+1, &fdsetR, NULL,  
          globalDataPtr->comData.fdTAb[WatchdogTimerFd],  
          globalDataPtr->comData.fdTAb[WatchdogFd], DxTrue);
```

This function (`FD_ISSET`) checks if the bit corresponding to the file descriptor given as the first argument is set in the select mask for which the address is given as the second argument. If the bit is set, a one (true) is returned, otherwise zero (false).

```
if (FD_ISSET(globalDataPtr->comData.fdTAb[UartFd],  
             &fdsetR))  
{  
    globalDataPtr->state = UartException;  
}  
else if (FD_ISSET(  
         globalDataPtr->comData.fdTAb[DownStreamCmndFd],  
         &fdsetR))  
{  
    /* Cancel idle read descriptor */
```

This ioctl function (MviCReadCancel) deactivates a Read Descriptor that has been previously activated. If the Read Descriptor is cancelled, DxTrue is returned in the third argument. If DxFalse is returned, the application has been awakened on the Read Descriptor after leaving select and must remove the selection condition by calling select again without putting out a new Read Descriptor. DxFalse is also returned if a Read Descriptor is cancelled twice, which is not allowed and is considered a programming error.

```
        if (ioctl(globalDataPtr->comData.fdTab[UartFd],
                  MviCReadCancel, &readDescIsCanceled)
            == ERROR)
        {
            /* System Message */
        }
        globalDataPtr->state = DownStreamCmd;
    }
    break;

case DownStreamCmd:
    handleDownStreamCmd(dlPtr, globalDataPtr);
    break;

case IdleTrafficPoll:
    handleIdleTrafficPoll(dlPtr, globalDataPtr);
    break;

case MessageFirstPart:
    handleMessagePart1FromSlave(dlPtr, globalDataPtr);
    break;

case MessageFinalPart:
    handleMessagePart2FromSlave(dlPtr, globalDataPtr);
    break;

case CharTimeout:
    handleCharacterTimeoutInMaster(dlPtr, globalDataPtr);
    break;

case UartException:
    handleUARTException(dlPtr, globalDataPtr);
    break;
    }
}
}
```

handleDownStreamCmd()

```
void handleDownStreamCmd(MviDL* dlPtr, XYGlobalData*
                        globalDataPtr)
{
    MviBuffer      tempBuffer;
    u_int16        chkSumTemp;
    int16          dataLength;
    int             turnTime;
    int             txTime;
    int             width;
    MviFdSet        fdsetR;
    MviFdSet        fdsetW;

    globalDataPtr->typeOfCmnd = DnStreamCommand;
    if (globalDataPtr->firstTXAttempt)
    {
```

This function (mviGetStreamCmnd) fetches a stream command from a FIFO, in this case the DownStreamCmnd FIFO. The Protocol Stub sends a downstream command to the Protocol Handler using the function mvDownStreamCmnd(). The Protocol Handler fetches the command using the function mviGetStreamCmnd, handles the command and responds to the Protocol Stub using the function mviDownStreamResp(). The Protocol Stub fetches the response using the function mviGetStreamResp(), which concludes the handshaking between the Protocol Handler and the Protocol Stub.

```
        if (mviGetStreamCmnd(
            globalDataPtr->comData.fdTAb[DownStreamCmndFd],
            &globalDataPtr->comData.recvCmndPtr) != MviOk)
        {
            /* System message sent in mviGetStreamCmnd() */
        }
    }
    switch (globalDataPtr->comData.recvCmndPtr->cmndCode)
    {
        case MviRestartHndl:
        {
            /*-----
            Send a global disconnect command to the stub
            (node number = 0).
            -----*/
```

This function (mviSendLinkStatus) sends a link status command to the Protocol Stub. The Protocol Stub interprets the command and takes necessary actions, i.e., connects one node on the link or disconnects one or all nodes on the link.

```
        mviSendLinkStatus(dlPtr, (MviGlobalData*) globalDataPtr,  
                           MviSLinkDownAllNodes);  
  
        break;  
    }  
  
default:  
    {  
        /*-----  
        Fetch pointer to DL-entry  
        -----*/
```

This function (mviGetNodeDLEntry) fetches the pointer to the DL-entry for a given node number.

```
    if (mviGetNodeDLEntry(dlPtr,  
        globalDataPtr->comData.recvCmndPtr->node,  
        &globalDataPtr->comData.dLEntryPtr) != MviOk)  
    {  
        /* System message sent in mviGetNodeDLEntry() */  
    }  
  
    /* -----  
    Fetch pointer to Node Status  
    -----*/
```

This function (mviGetNodeStatusInfo) fetches the pointer to the Status Info struct for a given node number.

```
    if (mviGetNodeStatusInfo(dlPtr,  
        globalDataPtr->comData.recvCmndPtr->node,  
        &globalDataPtr->comData.nodeStatusPtr) != MviOk)  
    {  
        /* System message sent in mviGetNodeStatusInfo() */  
    }  
  
    /* -----  
    Fetch pointer to Node Diagnostics  
    -----*/
```

This function (mviGetNodeDiagInfo) fetches the pointer to the Diag Info struct for a given node number.

```
if (mviGetNodeDiagInfo(dlPtr,
    globalDataPtr->comData.recvCmdPtr->node,
    &globalDataPtr->comData.nodeDiagPtr) != MviOk)
{
    /* System message sent in mviGetNodeDiagInfo() */
}

/*-----
Map command from MviCmdInfo to XY format, convert
to ASCII format, calculate and add checksum
-----*/
if ((dataLength = mapPHI2XY(dlPtr->setupDataPtr,
    globalDataPtr->comData.recvCmdPtr,
    tempBuffer)) != -1)
{
    tempBuffer[dataLength]=mviLrc(tempBuffer,dataLength);
    dataLength++;
    dataLength = convBin2ASCII(tempBuffer,
        dlPtr->writeDataBufPtr->buffer[0], dataLength);
}
else
{
    /*Error handling */
    globalDataPtr->statusIsModified = DxTrue;
    globalDataPtr->state = Idle;
    return;
}
}
/*-----
Cancel idle traffic poll timer
-----*/
```

This ioctl function (MviCWDTCancel) stops the Watchdog Timer corresponding to the file descriptor given as the first argument.

```
if (ioctl(((XYDLEntry*)globalDataPtr->
          comData.dlEntryPtr)->pollTimerFd,
          MviCWDTCancel, 0) != OK)
{
    /*Error handling */
}

/*-----
Prepare write descriptor with the command, Start
TX timer, send the message to the slave and wait
for transmission completed
-----*/

dlPtr->writeDescriptorPtr->noOfChar = dataLength;
dlPtr->writeDescriptorPtr->writeBuffer =
    dlPtr->writeDataBufPtr->buffer[0];
```

This function (mviSendData) starts a write transaction by sending a write descriptor to the UART. When the transaction is started, the Protocol Handler enters select, waiting to be awakened by the MVI UART device. When awakened by the UART device, the Protocol Handler fetches the wake up reason using the ioctl function MviCGetTxWakeupReason and acts according to the reason. That concludes the write transaction.

```
if (mviSendData(globalDataPtr->comData.fdTab[UartFd],
                dlPtr->writeDescriptorPtr) == ERROR)
{
    /*Error handling */
}

/* Calculate the txTime */
```

This ioctl function (MviCWDTStart) starts the Watchdog Timer corresponding to the file descriptor given as the first argument.

```
if (ioctl(globalDataPtr->comData.fdTAb[TXTimerFd],
          MviCWDTStart, (int) &txTime) != OK)
{
    /*Error handling */
}

FD_ZERO(&fdsetR);
FD_ZERO(&fdsetW);
width = (globalDataPtr->comData.fdTAb[TXTimerFd] >
         globalDataPtr->comData.fdTAb[UartFd]) ?
         globalDataPtr->comData.fdTAb[TXTimerFd] :
         globalDataPtr->comData.fdTAb[UartFd];
FD_SET(globalDataPtr->comData.fdTAb[TXTimerFd], &fdsetR);
FD_SET(globalDataPtr->comData.fdTAb[UartFd], &fdsetW);

mviSelect(width+1, &fdsetR, &fdsetW,
          globalDataPtr->comData.fdTAb[WatchdogTimerFd],
          globalDataPtr->comData.fdTAb[WatchdogFd], DxTrue);

if(FD_ISSET(globalDataPtr->comData.fdTAb[UartFd],
            &fdsetW))
{
    /*-----
    Resume reason is UartEvent: stop TXTimer, fetch
    wakeup reason, respond to stub if error and set
    state according to wakeup reason
    -----*/

    if (ioctl(globalDataPtr->comData.fdTAb[TXTimerFd],
              MviCWDTCancel, 0) != OK)
    {
        /*Error handling */
    }
    if (ioctl(globalDataPtr->comData.fdTAb[UartFd],
              MviCGetTxWakeupReason,
              (int) &globalDataPtr->comData.wakeupReason) !=OK)
    {
        /*Error handling */
    }
}
```

```
if (globalDataPtr->comData.wakeupReason &
    MviTXCompleted)
{
    /* Calculate turnTime */
    if (ioctl(globalDataPtr->comData.fdTab[AckTimerFd],
        MviCWDTStart, (int) &turnTime) != OK)
    {
        /*Error handling */
    }
    globalDataPtr->state = MessageFirstPart;
}
else
{
    globalDataPtr->comData.recvCmndPtr->rspCode =
        MviCRspHWEError;
```

This function (mviDownStreamResp) sends a response on a mviDownStreamCmnd. This must always be done since the Protocol Stub expects response on a command to the Protocol Handler.

```
        if (mviDownStreamResp(globalDataPtr->
            comData.fdTAb[DownStreamRespFd],
            &globalDataPtr->comData.recvCmndPtr) != MviOk)
        {
            /* System message sent in mviDownStreamResp() */
        }
        globalDataPtr->state = UartException;
    }
}
else if (FD_ISSET(globalDataPtr->
    comData.fdTAb[TXTimerFd], &fdsetR))
{
    /*-----
    Resume reason is TXTimeout: Increment error
    counter in MviDiagInfo, respond to stub with
    error and set state to UartException
    -----*/

    dlPtr->diagInfoPtr->ctrlErr++;
    globalDataPtr->comData.recvCmndPtr->rspCode =
        MviCRspHWEError;
    if (mviDownStreamResp(globalDataPtr->
        comData.fdTAb[DownStreamRespFd],
        &globalDataPtr->comData.recvCmndPtr) != MviOk)
    {
        /* System message sent in mviDownStreamResp() */
    }
    globalDataPtr->txTimedOut = DxTrue;
    globalDataPtr->state = UartException;
}
else
{
    /*Error handling */
}
}
}
```

handleMessagePart1FromSlave()

```
void handleMessagePart1FromSlave(MviDL* dlPtr, XYGlobalData*
                                globalDataPtr)
{
    int          pollTime;
    int          width;
    u_int16      dataLength;
    MviFdSet     fdsetR;

    /*-----
       Set trig condition in read descriptor to trig on colon
       + 4 characters
       -----*/
    dlPtr->readDescriptorPtr->modeOfOperation = 0;
    dlPtr->readDescriptorPtr->maxBufLength = MviBufSize;
    dlPtr->readDescriptorPtr->postTrigLength = 4;
    dlPtr->readDescriptorPtr->readBuffer =
        dlPtr->readDataBufPtr->buffer[0];
    dlPtr->readDescriptorPtr->CCOpt = 0;
    dlPtr->readDescriptorPtr->noOfCC = 1;
    dlPtr->readDescriptorPtr->CCSequence[0] = ':';

    /*-----
       Start a read transaction, wait for and handle event
       -----*/
```

This function (mviReqData) starts a read transaction by sending down a Read Descriptor to the MVI UART device. When the transaction is started, the Protocol Handler enters select, waiting to be awakened by the MVI UART device. When awakened by the UART device, the Protocol Handler fetches the wake up reason using the ioctl function MviCGetRxWakeupReason and acts according to the reason. That concludes the read transaction.

```
if (mviReqData(globalDataPtr->comData.fdTab[UartFd],
    dlPtr->readDescriptorPtr) == ERROR)
{
    /*Error handling */
}
FD_ZERO(&fdsetR);
width = (globalDataPtr->comData.fdTab[AckTimerFd] >
    globalDataPtr->comData.fdTab[UartFd]) ?
    globalDataPtr->comData.fdTab[AckTimerFd] :
    globalDataPtr->comData.fdTab[UartFd];
FD_SET(globalDataPtr->comData.fdTab[AckTimerFd], &fdsetR);
FD_SET(globalDataPtr->comData.fdTab[UartFd], &fdsetR);

mviSelect(width+1, &fdsetR, NULL,
    globalDataPtr->comData.fdTab[WatchdogTimerFd],
    globalDataPtr->comData.fdTab[WatchdogFd], DxTrue);

if (FD_ISSET(globalDataPtr->comData.fdTab[UartFd], &fdsetR))
{
    /*-----
    Resume Reason is UartEvent: Fetch wakeup reason and
    act accordingly
    -----*/
}
```

This ioctl command (MviCGetRxWakeupReason) fetches the reason why the application was awakened from select.

```
if (ioctl(globalDataPtr->comData.fdTab[UartFd],
    MviCGetRxWakeupReason,
    (int) &globalDataPtr->comData.wakeupReason) != OK)
{
    /*Error handling */
}
if (globalDataPtr->comData.wakeupReason & MviRXCompleted)
{
    /*-----
    Wakeup reason is RXCompleted: Convert address and
    func code to binary. If address and func code is as
    expected, set state to MessageFinalPart, otherwise
    set error bit in MviStatusInfo, respond with error
    to stub and restart idle traffic poll timer
    -----*/
}
```

This function (mviGetData) ends the read transaction by putting the data into the Read Descriptor.

```
    if (mviGetData(globalDataPtr->comData.fdTab[UartFd],
        dlPtr->readDescriptorPtr) == ERROR)
    {
        /*Error handling */
    }
    dataLength = convASCII2Bin(dlPtr->
        readDescriptorPtr->readBuffer,
        dlPtr->readDescriptorPtr->readBuffer,
        dlPtr->readDescriptorPtr->noOfReceivedChars,
        DxFalse, DxFalse);
    dlPtr->readDescriptorPtr->noOfReceivedChars=dataLength;
    globalDataPtr->state = MessageFinalPart;
}
else if (globalDataPtr->comData.wakeupReason
        &MviCharTimeout)
{
    /*-----
    Wakeup reason is CharTimeout: Increment error
    counter in MviDiagInfo and set state to CharTimeout
    -----*/
    globalDataPtr->comData.nodeDiagPtr->otherErr++;
    globalDataPtr->state = CharTimeout;
}
else
{
    /*-----
    Wakeup Reason is not RXCompleted or CharTimeout:
    Respond with error to stub and set state to
    UartException
    -----*/
    if (mviDownStreamResp(globalDataPtr->
        comData.fdTab[DownStreamRespFd],
        &globalDataPtr->comData.recvCmndPtr) != MviOk)
    {
        /* System message sent from mviDownStreamResp() */
    }
    globalDataPtr->state = UartException;
}
}
```

handleMessagePart2FromSlave()

```
void handleMessagePart2FromSlave(MviDL* dlPtr,
                                XYGlobalData* globalDataPtr)
{
    int          width;
    MviFdSet      fdsetR;

    /*-----
       Set trig condition in read descriptor to trig on CR + LF
    -----*/
    dlPtr->readDescriptorPtr->modeOfOperation = MviRegard;
    dlPtr->readDescriptorPtr->maxBufLength = MviBufSize -
        dlPtr->readDescriptorPtr->noOfReceivedChars;
    dlPtr->readDescriptorPtr->postTrigLength = 0;
    dlPtr->readDescriptorPtr->readBuffer +=
        dlPtr->readDescriptorPtr->noOfReceivedChars;
    dlPtr->readDescriptorPtr->CCOpt = 0;
    dlPtr->readDescriptorPtr->noOfCC = 2;
    dlPtr->readDescriptorPtr->CCSequence[0] = '\r';
    dlPtr->readDescriptorPtr->CCSequence[1] = '\n';
    /*-----
       Start a read transaction, wait for and handle event
    -----*/
    if (mviReqData(globalDataPtr->comData.fdTab[UartFd],
        dlPtr->readDescriptorPtr) == ERROR)
    {
        /*Error handling */
    }
    FD_ZERO(&fdsetR);
    width = (globalDataPtr->comData.fdTab[AckTimerFd] >
        globalDataPtr->comData.fdTab[UartFd]) ?
        globalDataPtr->comData.fdTab[AckTimerFd] :
        globalDataPtr->comData.fdTab[UartFd];
    FD_SET(globalDataPtr->comData.fdTab[AckTimerFd], &fdsetR);
    FD_SET(globalDataPtr->comData.fdTab[UartFd], &fdsetR);

    mviSelect(width+1, &fdsetR, NULL,
        globalDataPtr->comData.fdTab[WatchdogTimerFd],
        globalDataPtr->comData.fdTab[WatchdogFd], DxTrue);

    if (FD_ISSET(globalDataPtr->comData.fdTab[UartFd], &fdsetR))
    {
        /*-----
           Resume Reason is UartEvent: Fetch wakeup reason and
           act accordingly
        -----*/
    }
}
```

```
if (ioctl(globalDataPtr->comData.fdTab[UartFd],
        MviCGetRxWakeupReason,
        (int) &globalDataPtr->comData.wakeupReason) != OK)
{
    /*Error handling */
}
if (globalDataPtr->comData.wakeupReason & MviRXCompleted)
{
    /*-----
    Wakeup reason is RXCompleted: Handle response
    message from slave
    -----*/
    handleResponseFromSlave(dlPtr, globalDataPtr);
}
else if (globalDataPtr->comData.wakeupReason &
        MviCharTimeout)
{
    /*-----
    Wakeup reason is CharTimeout: Increment error
    counter in MviDiagInfo and set state to CharTimeout
    -----*/
    globalDataPtr->comData.nodeDiagPtr->otherErr++;
    globalDataPtr->state = CharTimeout;
}
else
{
    /*-----
    Wakeup Reason is not RXCompleted or CharTimeout:
    Respond with error to stub and set state to
    UartException
    -----*/
    if (globalDataPtr->typeOfCmd == DnStreamCommand)
    {
        globalDataPtr->comData.recvCmdPtr->rspCode =
            MviCRspHwError;
        if (mviDownStreamResp(globalDataPtr->
            comData.fdTab[DownStreamRespFd],
            &globalDataPtr->comData.recvCmdPtr) != MviOk)
        {
            /* System message sent from mviDownStreamResp() */
        }
    }
}
```

```
else if (FD_ISSET(globalDataPtr->
                                comData.fdTab[AckTimerFd], &fdsetR))
{
    /*-----
    Resume Reason is AckTimeout: Call
    handleRetransmInMaster
    -----*/
    handleRetransmInMaster(dlPtr, globalDataPtr);
}
else
{
    /*Error handling */
}
}
```

handleResponseFromSlave()

```
void handleResponseFromSlave(MviDL* dlPtr, XYGlobalData*
                                globalDataPtr)
{
    int            pollTime;
    u_int16        dataLength;
    u_int16        chkSumCRC16;
    u_int16        chkSumTemp;
    u_int8         chkSumLRC;
    DxBoolean      chkSumIsOk;

    /* Stop AckTimeout Timer */

    if (ioctl(globalDataPtr->comData.fdTab[AckTimerFd],
               MviCWDTCancel, 0) != OK)
    {
        /*Error handling */
    }
    /*-----
    Convert second part of response message to binary (RTU)
    format, adjust read buffer pointer and noOfReceivedChars
    to whole message and check if checksum is correct
    -----*/
    dataLength = convASCII2Bin(dlPtr->
                               readDescriptorPtr->readBuffer,
                               dlPtr->readDescriptorPtr->readBuffer,
                               dlPtr->readDescriptorPtr->noOfReceivedChars,
                               DxFalse, DxFalse);
    dlPtr->readDescriptorPtr->readBuffer = dlPtr->
                                         readDataBufPtr->buffer[0];
    dlPtr->readDescriptorPtr->noOfReceivedChars = dataLength + 2;
    chkSumLRC = mviLrc(dlPtr->readDescriptorPtr->readBuffer,
                      dlPtr->readDescriptorPtr->noOfReceivedChars - 1);
    chkSumIsOk = (chkSumLRC == dlPtr->readDescriptorPtr->
                  readBuffer[dlPtr->readDescriptorPtr->
                              noOfReceivedChars - 1]) ? DxTrue : DxFalse;
```

```
if (chkSumIsOk)
{
switch (dlPtr->readDescriptorPtr->readBuffer[1])
{
case ReadCommand:
/*-----
Map message in XY buffer format to PHI
format and send it to the protocol stub
-----*/
mapXY2PHI(dlPtr,
dlPtr->readDescriptorPtr->readBuffer,
globalDataPtr->comData.recvCmndPtr);
globalDataPtr->
comData.recvCmndPtr->rspCode = MviCRspOk;
if (mviDownStreamResp(globalDataPtr->
comData.fdTAb[DownStreamRespFd],
&globalDataPtr->comData.recvCmndPtr) != MviOk)
{
/* System message sent from mviDownStreamResp() */
}
break;

case WriteCommand:
/*-----
Respond to the stub with MviCRspOk
-----*/
globalDataPtr->
comData.recvCmndPtr->rspCode = MviCRspOk;
if (mviDownStreamResp(globalDataPtr->
comData.fdTAb[DownStreamRespFd],
&globalDataPtr->comData.recvCmndPtr) != MviOk)
{
/* System message sent from mviDownStreamResp() */
}
break;

case LoopBackDiagTest:
/*-----
If Link Status bit in XYDLEntry is cleared, send
message "node is up" to stub and set Link Status
bit in XYDLEntry
-----*/
}
```

```
        if ((globalDataPtr->
            comData.dlEntryPtr->linkStatus & MviLinkUp) == 0)
        {
            mviSendLinkStatus(dlPtr,
                (MviGlobalData*) globalDataPtr, MviSLinkUp);
            globalDataPtr->
                comData.dlEntryPtr->linkStatus |= MviLinkUp;
        }
        break;

    default:
        /*Error handling */
        break;
    }
    /*-----
    Restart Idle Traffic Poll timer and set state to Idle
    -----*/
    pollTime = dlPtr->setupDataPtr->pollTout * 1000;
    if (ioctl(((XYDLEntry*) globalDataPtr->
        comData.dlEntryPtr)->pollTimerFd,
        MviCWDtStart, (int) &pollTime) != OK)
    {
        /*Error handling */
    }
    globalDataPtr->state = Idle;
}
else
{
    /*-----
    Checksum of message is not correct: increment counter
    in MviDiagInfo and handle retransmission using
    handleRetransmInMaster.
    -----*/
    globalDataPtr->comData.nodeDiagPtr->otherErr++;
    handleRetransmInMaster(dlPtr, globalDataPtr);
}
}
```

Appendix B DlkMVI Demo Protocol Configuration

B.1 MVI Demo Protocol Configuration and Application Building for Advant Controller 400

B.1.1 MS for MVI Demo Protocol Configuration

This section describes the Configuration MS used to configure the MVI Demo Protocol. These MS contain setup information for the CI535's hardware and software. Status and Diagnostics information is also received from the MVI Demo Protocol. The Status and Diagnostics information can be used by the application program.

B.1.1.1 Line Characteristics MS

Line characteristics are specified in one MS for each asynchronous communication port, which the MVI Demo Protocol reads during its start-up/restart and start-up of Advant Controller 400. The application program can also initiate a port by sending Line Characteristics MS with the SENDREQ PC element. For general information, see [Section 3.6.7.1, MS for MVI Protocol Configuration](#).

		MSn/MSn	
		MVI Data Set	
		(298.n)	
		Base part	
MSn	_____	1	NAME VALID 11
1	_____	2	ACT ERR 17
1/11	_____	3	IDENT
0	_____	4	NO_BREC
0	_____	5	NO_INT
23	_____	6	NO_INTL
0	_____	7	NO_REAL
3	_____	9	USER
SEND	_____	10	SOURCE
1	_____	12	BLOCKED
"submodule number"	_____	13	NET
-3	_____	14	NODE
1	_____	15	SCAN_FTR
YES	_____	18	SORT_REF

Figure B-1. MVI Data Set for Line Characteristics Base Part

			Value references
type of RTU	_____	8(1)	REF1
master/slave	_____	8(2)	REF2
bitrate	_____	8(3)	REF3
char. length	_____	8(4)	REF4
stopbits	_____	8(5)	REF5
parity	_____	8(6)	REF6
duplex	_____	8(7)	REF7
dial	_____	8(8)	REF8
pre-idle time	_____	8(9)	REF9
carrier delay	_____	8(10)	REF10
char. time-out	_____	8(11)	REF11
turn-around time	_____	8(12)	REF12
UART time-out	_____	8(13)	REF13
retransmissions	_____	8(14)	REF14
poll cycle time	_____	8(15)	REF15
parameter 1	_____	8(16)	REF16
parameter 2	_____	8(17)	REF17
parameter 3	_____	8(18)	REF18
parameter 4	_____	8(19)	REF19
parameter 5	_____	8(20)	REF20
parameter 6	_____	8(21)	REF21
parameter 7	_____	8(22)	REF22
parameter 8	_____	8(23)	REF23

Figure B-2. MVI Data Set for Line Characteristics Reference Part

The parameters REF1 - REF23 are stored in the DAT elements referenced by the Line Characteristics MS. You can give DAT elements arbitrary names.

The parameters have the following significance:

Table B-1. Description of MS for Line Characteristics

Terminal	Parameter Name	Recommended Value (Min-Max)	Description
NAME			Arbitrary name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT		1 or 11	Identity of the MVI Data Set 1 for port 1 11 for port 2
NO_BREC		0	
NO_INT		0	
NO_INTL		23	Number of Integer Long DAT references
NO_REAL		0	
USER		3	
SOURCE		SEND	
BLOCKED		1	Note: The MS must be blocked. MS requested by the CI535 module at start-up.
NET		7 - 11	CI535 submodule number. Record number for the CI535 data base element for the MVI Demo Protocol port defines the slave module number. Record No. 1 => submodule = 7 Record No. 2 => submodule = 8 Record No. 3 => submodule = 9 (only valid for AC 450) Record No. 4 => submodule = 10 (only valid for AC 450) Record No. 5 => submodule = 11 (only valid for AC 450)
NODE		-3	Always -3 for Configuration MS
SCAN_FTR		1	Without function when BLOCKED=1
SORT_REF		YES	REF1 - 24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID			1 after first successful transmission. Reset by user.
ERR			1 if the transmission has failed due to lost contact with destination node or queue full to CI535
REF1	Type of RTU	20	Protocol type

Table B-1. Description of MS for Line Characteristics (Continued)

Terminal	Parameter Name	Recommended Value (Min-Max)	Description
REF2	Slave/Master	0 or 1	0 = The port is a Slave 1 = The port is a Master
REF3	Bitrate		Transmission speed: 150, 300, 600, 1200, 2400, 4800, 9600 or 19200 bits/s.
REF4	Character Length	7	The number of bits/character
REF5	Stopbits	10	5 = 0.5 Stop bits 10 = 1 Stop bit 15 = 1.5 Stop bits 20 = 2 Stop bits
REF6	Parity	2	0 = No parity 1 = Odd parity 2 = Even parity
REF7	Duplex	1	0 = Half duplex 1 = Full duplex 2 = Half duplex and ignore DCD
REF 8	Dial	0	Dialing. 0 = disabled. Not supported by the MVI Demo Protocol.
REF9	Pre-idle time	3 for master 0 for slave (0 - 255)	Pre-idle time in number of character transmission times. Time to allow the carrier wave to stabilize before transmission of the first character. Restrictions in half duplex: “(own) pre-idle. time >= (opposite side) post-idle time” See also Figure B-3 below.
REF10	Post-idle time	1 half duplex 0 full duplex (0 - 255)	Post-idle time in number of character transmission times. Time to wait after transmission of the last character before deactivating RTS. This delay is used to avoid destruction of the last character in the message, due to lost carrier. See also Figure B-3 below.
REF11	Char Time-out	3 (0 - 255)	The number of character transmission times to wait until the message is considered interrupted.

Table B-1. Description of MS for Line Characteristics (Continued)

Terminal	Parameter Name	Recommended Value (Min-Max)	Description
REF12	Turn-around time	100 (5 - 10000)	Time in milliseconds to wait from the last character in the command until the first character in the reply, that is time-out time, where the master waits for a response from the slave node. Note: This time is dependent on the times defined in REF8 and REF9 (or corresponding set-up) in the slave node and must be adjusted accordingly. You must also include delays that may occur in the slave units. See also Figure B-4 below.
REF 13	UART time-out	2 for half duplex 1 for full duplex (0 - 255)	UART transmission time-out in number of character transmission times. If half duplex is used, a check of DCD will be made by the send routine before activating RTS. If DCD is activated the send routine will wait for DCD to be deactivated up to this time before aborting the transmission..
REF14	Retransmissions	2 for Master 0 for Slave (0 - 200)	Max. No. of retransmissions before the line is considered broken.
REF15	Poll cycle time	96000/Bitrate for Master 1.2*96000/Bitrate for Slave (5 - 770)	Max. allowed time in seconds between two polling cycles. In slave mode, the slave disconnects the link when this time between polls is exceeded.
REF 16	Parameter 1		Not used by the MVI Demo Protocol.
REF 17	Parameter 2		Not used by the MVI Demo Protocol.
REF 18	Parameter 3		Not used by the MVI Demo Protocol.
REF 19	Parameter 4		Not used by the MVI Demo Protocol.
REF 20	Parameter 5		Not used by the MVI Demo Protocol.
REF 21	Parameter 6		Not used by the MVI Demo Protocol.
REF 22	Parameter 7		Not used by the MVI Demo Protocol.
REF 23	Parameter 8		Not used by the MVI Demo Protocol.

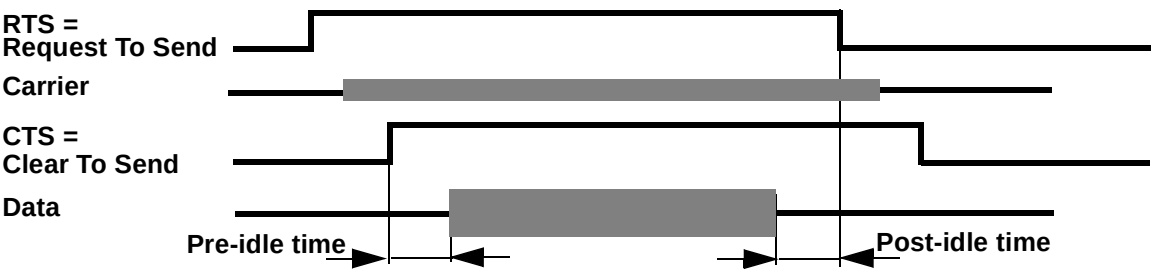


Figure B-3. Pre-idle Time and Post-idle Time in Half Duplex Mode

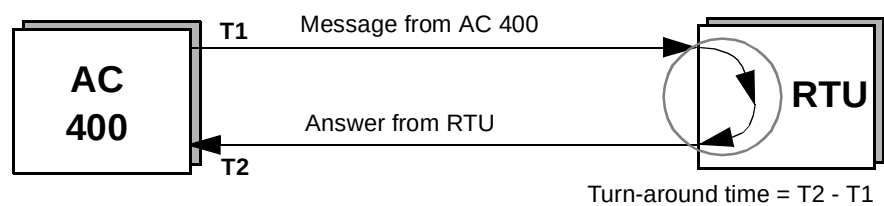


Figure B-4. Turn-around Time

B.1.1.2 Network Configuration MS

The node numbers for the RTUs are defined in one MS for each communication port. For general information, see [Section 3.6.7.1, MS for MVI Protocol Configuration](#).

MSn/MSn				MVI Data Set (298.n)	
				Base part	
MSn	_____	1	NAME	VALID	11 —
1	_____	2	ACT	ERR	17 —
2/12	_____	3	IDENT		
0	_____	4	NO_BREC		
0	_____	5	NO_INT		
24	_____	6	NO_INTL		
0	_____	7	NO_REAL		
3	_____	9	USER		
SEND	_____	10	SOURCE		
1	_____	12	BLOCKED		
"submodule number"	_____	13	NET		
-3	_____	14	NODE		
1	_____	15	SCAN_FTR		
YES	_____	18	SORT_REF		
				Value references	
RTU-number for RTU1	_____	8(1)	REF1		
0	_____	8(2)	REF2		
0	_____	8(3)	REF3		
RTU-number for RTU2	_____	8(4)	REF4		
0	_____	8(5)	REF5		
0	_____	8(6)	REF6		
RTU-number for RTU3	_____	8(7)	REF7		
0	_____	8(8)	REF8		
0	_____	8(9)	REF9		
RTU-number for RTU4	_____	8(10)	REF10		
0	_____	8(11)	REF11		
0	_____	8(12)	REF12		
RTU-number for RTU5	_____	8(13)	REF13		
0	_____	8(14)	REF14		
0	_____	8(15)	REF15		
RTU-number for RTU6	_____	8(16)	REF16		
0	_____	8(17)	REF17		
0	_____	8(18)	REF18		
RTU-number for RTU7	_____	8(19)	REF19		
0	_____	8(20)	REF20		
0	_____	8(21)	REF21		
RTU-number for RTU8	_____	8(22)	REF22		
0	_____	8(23)	REF23		
0	_____	8(24)	REF24		

Figure B-5. Example of MVI Data Set for Network Configuration

The parameters REF1 - REF24 are stored in DAT elements referenced by the Network Configuration MS. You can give the DAT elements arbitrary names.

The parameters have the following significance:

Table B-2. Description of MS for Network Configuration

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT		2 or 12	Identity of the MVI Data Set 2 for port 1 12 for port 2
NO_BREC		0	
NO_INT		0	
NO_INTL		24	Each defined RTU requires 3 Integer Long DATs.
NO_REAL		0	
USER		3	
SOURCE		SEND	
BLOCKED		1	Note: The MS must be blocked. MS requested by the CI535 module at startup.
NET		7 - 11	CI535 submodule number. Record number for the CI535 data base element for the MVI Demo Protocol port defines the slave module number. Record No. 1 => submodule = 7 Record No. 2 => submodule = 8 Record No. 3 => submodule = 9 (only valid for AC 450) Record No. 4 => submodule = 10 (only valid for AC 450) Record No. 5 => submodule = 11 (only valid for AC 450)
NODE		-3	Always -3 for Configuration MS
SCAN_FTR		1	Without function when BLOCKED = 1
SORT_REF		YES	REF1-24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID			1 after first successful transmission. Reset by user.
ERR			1 if the transmission has failed due to lost contact with destination node or queue full to CI535.

Table B-2. Description of MS for Network Configuration (Continued)

Terminal	Parameter Name	Recommended Value	Description
REF1 REF4 REF7 REF10 REF13 REF16 REF19 REF22	RTU number: RTU1 RTU2 RTU3 RTU4 RTU5 RTU6 RTU7 RTU8	1 - 99	Node number of: RTU1 RTU2 RTU3 RTU4 RTU5 RTU6 RTU7 RTU8 NOTE: All the defined node numbers must be unique within the connected controller.
REF2 REF5 REF8 REF11 REF14 REF17 REF20 REF23	Area number: RTU1 RTU2 RTU3 RTU4 RTU5 RTU6 RTU7 RTU8	0	Area code: Telephony is not supported by the MVI Demo Protocol, so these REF values are set to 0 (zero).
REF3 REF6 REF9 REF12 REF15 REF18 REF21 REF24	Phone number: RTU1 RTU2 RTU3 RTU4 RTU5 RTU6 RTU7 RTU8	0	Subscriber number: Telephony is not supported by the MVI Demo Protocol, so these REF values are set to 0 (zero).

B.1.1.3 RTU Status MS

The status of every RTU is stored in an MS for each port. The status word is updated by the MVI Demo Protocol and **may only be read by the PC program**. For general information, see [Section 3.6.7.1, MS for MVI Protocol Configuration](#).

			MSn/MSn	
			MVI Data Set	
			(298.n)	
			Base part	
MSn	_____	1	NAME	VALID 11 —
1	_____	2	ACT	ERR 17 —
3/13	_____	3	IDENT	
8	_____	4	NO_BREC	
0	_____	5	NO_INT	
16	_____	6	NO_INTL	
0	_____	7	NO_REAL	
3	_____	9	USER	
RECEIVE	_____	10	SOURCE	
0	_____	12	BLOCKED	
"submodule number"	_____	13	NET	
-3	_____	14	NODE	
1	_____	15	SCAN_FTR	
YES	_____	18	SORT_REF	
			Value references	
status1 RTU1	_____	8(1)	REF1	
status1 RTU2	_____	8(2)	REF2	
status1 RTU3	_____	8(3)	REF3	
status1 RTU4	_____	8(4)	REF4	
status1 RTU5	_____	8(5)	REF5	
status1 RTU6	_____	8(6)	REF6	
status1 RTU7	_____	8(7)	REF7	
status1 RTU8	_____	8(8)	REF8	
status2 RTU1	_____	8(9)	REF9	
status2 RTU2	_____	8(10)	REF10	
status2 RTU3	_____	8(11)	REF11	
status2 RTU4	_____	8(12)	REF12	
status2 RTU5	_____	8(13)	REF13	
status2 RTU6	_____	8(14)	REF14	
status2 RTU7	_____	8(15)	REF15	
status2 RTU8	_____	8(16)	REF16	
status3 RTU1	_____	8(17)	REF17	
status3 RTU2	_____	8(18)	REF18	
status3 RTU3	_____	8(19)	REF19	
status3 RTU4	_____	8(20)	REF20	
status3 RTU5	_____	8(21)	REF21	
status3 RTU6	_____	8(22)	REF22	
status3 RTU7	_____	8(23)	REF23	
status3 RTU8	_____	8(24)	REF24	

Figure B-6. MVI Data Set for RTU Status

The parameters REF1 - REF24 are stored in DAT elements referenced from this Status MS.
You can give the DAT elements arbitrary names.

The parameters have the following significance:

Table B-3. Description of MS for RTU Status

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT		3 or 13	Identity of the MVI Data Set 3 for port 1 13 for port 2
NO_BREC		8	Number of Boolean DAT references
NO_INT		0	
NO_INTL		16	Number of Integer Long DAT references
NO_REAL		0	
USER		3	
SOURCE		RECEIVE	
BLOCKED		0	Note: The MS must not be blocked.
NET		7 - 11	CI535 submodule number. Record number for the CI535 data base element for the MVI Demo Protocol port defines the slave module number. Record no 1 => submodule = 7 Record no 2 => submodule = 8 Record no 3 => submodule = 9 (only valid for AC 450) Record no 4 => submodule = 10 (only valid for AC 450) Record no 5 => submodule = 11 (only valid for AC 450)
NODE		-3	Always -3 for Configuration MS.
SCAN_FTR		1	Used for reset of the VALID flag. VALID is set to 0 when the Time = 3*SCAN_FTR * MS_SCANTIME has passed. Default value for MS_SCANTIME = 2 s.
SORT_REF		YES	REF1 - 24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID			1 when the MS is updated. See also SCAN_FTR above.
ERR			1 if wrong data size is received.

Table B-3. Description of MS for RTU Status (Continued)

Terminal	Parameter Name	Recommended Value	Description
REF1	Status1 RTU1		First status word for RTU1
REF2	Status1 RTU2		First status word for RTU2
REF3	Status1 RTU3		First status word for RTU3
REF4	Status1 RTU4		First status word for RTU4
REF5	Status1 RTU5		First status word for RTU5
REF6	Status1 RTU6		First status word for RTU6
REF7	Status1 RTU7		First status word for RTU7
REF8	Status1 RTU8		First status word for RTU8
REF9	Status2 RTU1		Second status word for RTU1
REF10	Status2 RTU2		Second status word for RTU2
REF11	Status2 RTU3		Second status word for RTU3
REF12	Status2 RTU4		Second status word for RTU4
REF13	Status2 RTU5		Second status word for RTU5
REF14	Status2 RTU6		Second status word for RTU6
REF15	Status2 RTU7		Second status word for RTU7
REF16	Status2 RTU8		Second status word for RTU8
REF17	Status3 RTU1		Third status word for RTU1
REF18	Status3 RTU2		Third status word for RTU2
REF19	Status3 RTU3		Third status word for RTU3
REF20	Status3 RTU4		Third status word for RTU4
REF21	Status3 RTU5		Third status word for RTU5
REF22	Status3 RTU6		Third status word for RTU6
REF23	Status3 RTU7		Third status word for RTU7
REF24	Status3 RTU8		Third status word for RTU8

NOTE

You **must** build this MS with 24 DATs, according to the description above.

Status1 RTU1 - RTU8 (Boolean)

Contains information required for flow control, among other things. See [Table B-4](#) below.

Table B-4. Status1 RTU1 - RTU8 (Boolean)

Status1	Description	Valid modes
VALUE (bit0)	Link status The corresponding RTU reachable. Set to "1" when the remote node is reachable. Set to "0" when contact with remote node is lost (including restart of CI535).	Slave and Master
VALUE2 (bit1)	Reserved	
VALUE3 (bit2)	Reserved	
VALUE4 (bit3)	Illegal MS number received Set to "1" when illegal MS number is received from an RTU or from a AMPL application command. Set to "0" when receiving valid MS or command.	Slave and Master
VALUE5 (bit4)	Illegal function code received Set to "1" when illegal functions code is received from an RTU. Set to "0" when receiving valid MS.	Slave
VALUE6 (bit5)	Ready for Message Set to "1" when no command is pending, from the application to the RTU. Set to "0" when a command is pending, from the application to the RTU. Used by the PC program for flow control.	Master
VALUE7 (bit6)	Reserved	
VALUE8 (bit7)	Reserved	
VALUE9 (bit8)	Illegal function code reported from PLC. (MVI Demo Exception Response Code 1). The Function code can not be handled by the PLC. Set to "1" when response code 1 is received. Set to "0" when CI535 is restarted.	Master
VALUE10 (bit9)	Illegal data address reported from PLC. (MVI Demo Exception Response Code 2). The address referenced is not an allowable address for the addressed slave location. Set to "1" when response code 2 is received. Set to "0" when CI535 is restarted.	Master
VALUE11 (bit10) to VALUE32 (bit31)	User defined	Not used by the MVI Demo Protocol.

Status2 RTU1 - RTU8 (Integer Long)

If Status1 VALUE4 = 1 or VALUE10 = 1, that is, address error, the Status2 holds the address that caused the error. Status2 is reset at restart of the CI535.

Status3 RTU1 - RTU8 (Integer Long)

If Status1 VALUE5 = 1 or VALUE9 = 1, that is, illegal function code, the Status3 holds the function code that caused the error. Status3 is reset at restart of the CI535.

RTU Diagnostics Information MS

The diagnostics information of the UART Controller and every RTU is stored in an MS for each port. The diagnostics information is updated by the MVI Demo Protocol and **may only be read by the PC program.**

		MSn/MSn		MVI Data Set (298.n)	
				Base part	
MSn	_____	1	NAME	VALID	11 —
1	_____	2	ACT	ERR	17 —
254/255	_____	3	IDENT		
1	_____	4	NO_BREC		
0	_____	5	NO_INT		
17	_____	6	NO_INTL		
0	_____	7	NO_REAL		
3	_____	9	USER		
RECEIVE	_____	10	SOURCE		
0	_____	12	BLOCKED		
"submodule number"	_____	13	NET		
-3	_____	14	NODE		
1	_____	15	SCAN_FTR		
YES	_____	18	SOFT_REF		
				Value references	
controller status	_____	8(1)	REF1		
controller error	_____	8(2)	REF2		
parity error counter RTU1	_____	8(3)	REF3		
other errors counter RTU1	_____	8(4)	REF4		
parity error counter RTU2	_____	8(5)	REF5		
other errors counter RTU2	_____	8(6)	REF6		
parity error counter RTU3	_____	8(7)	REF7		
other errors counter RTU3	_____	8(8)	REF8		
parity error counter RTU4	_____	8(9)	REF9		
other errors counter RTU4	_____	8(10)	REF10		
parity error counter RTU5	_____	8(11)	REF11		
other errors counter RTU5	_____	8(12)	REF12		
parity error counter RTU6	_____	8(13)	REF13		
other errors counter RTU6	_____	8(14)	REF14		
parity error counter RTU7	_____	8(15)	REF15		
other errors counter RTU7	_____	8(16)	REF16		
parity error counter RTU8	_____	8(17)	REF17		
other errors counter RTU8	_____	8(18)	REF18		

Figure B-7. MVI Data Set for Diagnostics Information

Table B-5. Description of MS for Diagnostics Information

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT		254 or 255	Identity of the MVI Data Set 254 for port 1 255 for port 2
NO_BREC		1	Number of Boolean DAT references
NO_INT		0	
NO_INTL		17	Number of Integer Long DAT references
NO_REAL		0	
USER		3	
SOURCE		RECEIVE	
BLOCKED		0	Note: The MS must not be blocked.
NET		7 - 11	CI535 submodule number. Record number for the CI535 data base element for the MVI Demo Protocol port defines the slave module number. Record No. 1 => submodule = 7 Record No. 2 => submodule = 8 Record No. 3 => submodule = 9 (only valid for AC 450) Record No. 4 => submodule = 10 (only valid for AC 450) Record No. 5 => submodule = 11 (only valid for AC 450)
NODE		-3	Always -3 for Configuration MS.
SCAN_FTR		1	Used for reset of the VALID flag. VALID is set to 0 when the Time = 3*SCAN_FTR * MS_SCANTIME has passed. Default value for MS_SCANTIME = 2 s.
SORT_REF		YES	REF1 - 24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID			1 when the MS is updated. See also SCAN_FTR above.
ERR			1 if wrong data size is received.

Table B-5. Description of MS for Diagnostics Information (Continued)

Terminal	Parameter Name	Recommended Value	Description
REF1	Controller status		Current status of UART modem signals
REF2	Controller error		UART Controller errors counter
REF3	Parity error RTU1		Parity error word for RTU1
REF4	Other errors RTU1		Other errors counter for RTU1
REF5	Parity error RTU2		Parity error word for RTU2
REF6	Other errors RTU2		Other errors counter for RTU2
REF7	Parity error RTU3		Parity error word for RTU3
REF8	Other errors RTU3		Other errors counter for RTU3
REF9	Parity error RTU4		Parity error word for RTU4
REF10	Other errors RTU4		Other errors counter for RTU4
REF11	Parity error RTU5		Parity error word for RTU5
REF12	Other errors RTU5		Other errors counter for RTU5
REF13	Parity error RTU6		Parity error word for RTU6
REF14	Other errors RTU6		Other errors counter for RTU6
REF15	Parity error RTU7		Parity error word for RTU7
REF16	Other errors RTU7		Other errors counter for RTU7
REF17	Parity error RTU8		Parity error word for RTU8
REF18	Other errors RTU8		Other errors counter for RTU8

Table B-6. Controller Status (Boolean)

Status	Description
VALUE (bit0)	Current status of Data Set Ready (DSR)
VALUE2 (bit1)	Current status of Clear To Send (CTS)
VALUE3 (bit2)	Current status of Data Carrier Delay (DCD)
VALUE4 (bit3)	Current status of Ring Indicator (RI)
VALUE9 (bit8)	If set to 1, a hardware error was detected

B.1.1.4 Register Address MS

A translation between the Advant Controller 400 MS identities and the MVI Demo Protocol register addresses must be made, since the addressing of registers differs between Advant OCS and the MVI Demo Protocol.

For this purpose, a cross-reference table is used for the MVI Demo Protocol. This table is defined by a number of Register Address MS that must be built. These Register Address MS must contain the MVI Demo Protocol register addresses. For details, see [Section 3.6.7.1, MS for MVI Protocol Configuration](#).

First Register Address MS				Second Register Address MS			
MSn/MSn				MSn/MSn			
MVI Data Set (298.n)				MVI Data Set (298.n)			
Base part				Base part			
MSn — 1	NAME	VALID	11 —	MSn — 1	NAME	VALID	11 —
1 — 2	ACT	ERR	17 —	1 — 2	ACT	ERR	17 —
4/14 — 3	IDENT			5/15 — 3	IDENT		
0 — 4	NO_BREC			0 — 4	NO_BREC		
0 — 5	NO_INT			0 — 5	NO_INT		
24 — 6	NO_INTL			24 — 6	NO_INTL		
0 — 7	NO_REAL			0 — 7	NO_REAL		
3 — 9	USER			3 — 9	USER		
SEND — 10	SOURCE			SEND — 10	SOURCE		
1 — 12	BLOCKED			1 — 12	BLOCKED		
"submodule no." — 13	NET			"submodule no." — 13	NET		
-3 — 14	NODE			-3 — 14	NODE		
1 — 15	SCAN_FTR			1 — 15	SCAN_FTR		
YES — 18	SORT_REF			YES — 18	SORT_REF		
Value references				Value references			
Start reg. IDENT=1/101	REF1			Start reg. IDENT=25/125	REF1		
Start reg. IDENT=2/102	REF2			Start reg. IDENT=26/126	REF2		
Start reg. IDENT=3/103	REF3			Start reg. IDENT=27/127	REF3		
	:				:		
Start reg. IDENT=24/124	REF24			Start reg. IDENT=48/148	REF24		

Third Register Address MS				Fourth Register Address MS			
MSn/MSn				MSn/MSn			
MVI Data Set (298.n)				MVI Data Set (298.n)			
Base part				Base part			
MSn — 1	NAME	VALID	11 —	MSn — 1	NAME	VALID	11 —
1 — 2	ACT	ERR	17 —	1 — 2	ACT	ERR	17 —
6/16 — 3	IDENT			7/17 — 3	IDENT		
0 — 4	NO_BREC			0 — 4	NO_BREC		
0 — 5	NO_INT			0 — 5	NO_INT		
24 — 6	NO_INTL			24 — 6	NO_INTL		
0 — 7	NO_REAL			0 — 7	NO_REAL		
3 — 9	USER			3 — 9	USER		
SEND — 10	SOURCE			SEND — 10	SOURCE		
1 — 12	BLOCKED			1 — 12	BLOCKED		
"submodule no." — 13	NET			"submodule no." — 13	NET		
-3 — 14	NODE			-3 — 14	NODE		
1 — 15	SCAN_FTR			1 — 15	SCAN_FTR		
YES — 18	SORT_REF			YES — 18	SORT_REF		
Value references				Value references			
Start reg. IDENT=49/149	REF1			Start reg. IDENT=73/173	REF1		
Start reg. IDENT=50/150	REF2			Start reg. IDENT=74/174	REF2		
Start reg. IDENT=51/151	REF3			Start reg. IDENT=75/175	REF3		
	:				:		
Start reg. IDENT=72/172	REF24			Start reg. IDENT=96/196	REF24		

Figure B-8. MVI Data Sets for Register Addresses

The parameters REF1 - REF24 are stored in DAT elements referred to by this MVI Data Set. You can give the DAT elements arbitrary names.

The parameters have the following function:

Table B-7. Description of Register Address MS

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT		Port1: 4 - 7 Port2: 14 - 17	Identity of the MVI Data Set -First Register Address MS: Port 1: IDENT = 4 Port 2: IDENT = 14 -Second Register Address MS: Port 1: IDENT = 5 Port 2: IDENT = 15 -Third Register Address MS: Port 1: IDENT = 6 Port 2: IDENT = 16 -Fourth Register Address MS: Port 1: IDENT = 7 Port 2: IDENT = 17
NO_BREC		0	
NO_INT		0	
NO_INTL		24	Number of Integer Long DAT references.
NO_REAL		0	
USER		3	
SOURCE		SEND	
BLOCKED		1	Note: The MS must be blocked. MS requested by the CI535 module at start-up.
NET		7 - 11	CI535 submodule number. Record number for the CI535 data base element for the MVI Demo Protocol port defines the slave module number. Record No. 1 => submodule = 7 Record No. 2 => submodule = 8 Record No. 3 => submodule = 9 (only valid for AC 450) Record No. 4 => submodule = 10 (only valid for AC 450) Record No. 5 => submodule = 11 (only valid for AC 450)

Table B-7. Description of Register Address MS (Continued)

Terminal	Parameter Name	Recommended Value	Description
NODE		-3	
SCAN_FTR		1	Without function when BLOCKED=1.
VALID		1	1 after first successful transmission. Reset by user.
ERR			1 if the transmission has failed due to lost contact with destination node or queue full to CI535.
REF1 - REF24			Start register address for corresponding to: Sending Data MS with IDENT =1 - 96 and Receiving Data MS with IDENT = 101 - 196.

B.1.2 Command MS used in Master Mode

This section describes the Command MS used for control of the data transfer in master mode. The Command MS are only defined and used if the CI535 submodule is used as a master on the MVI Demo Protocol link.

These Command MS are used by the application program to activate read and write commands from CI535 to the corresponding RTU slave.

B.1.2.1 Control Commands

There are a number of control tasks to be done on the MVI Demo Protocol link. A number of Control Command MS are used for this purpose (IDENT=101-103 for port 1 and 111-113 for port 2).

The Control Command MS contain three DATs:

- The first DAT contains the start register address to read.
- The second DAT contains the number of registers (values) to read.
- The third DAT contains auxiliary information 1.

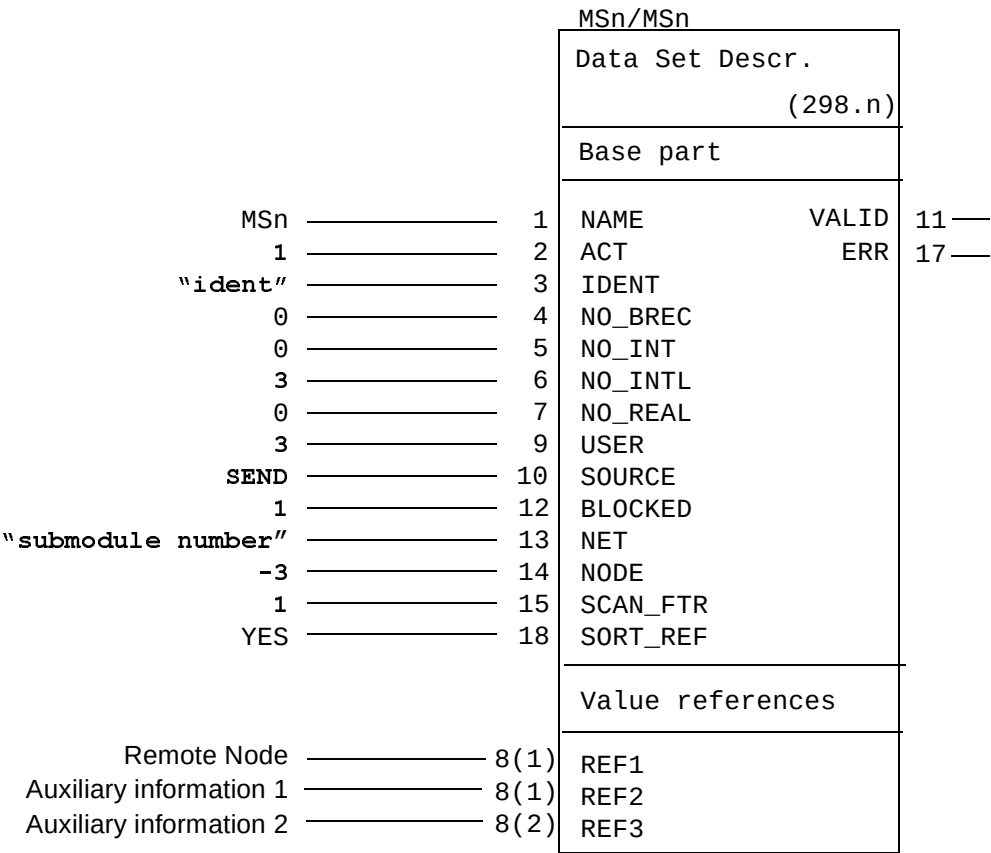


Figure B-9. Control Command MS

Table B-8. Description of MS for Control Commands

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT			The identity of the MVI Data Set identifies the command code: 101= Restart Handler Port 1 111 = Restart Handler Port 2 102 = Reset Diagnostics Port 1 112 = Reset Diagnostics Port 2 103 = Read Diagnostics Port 1 113 = Read Diagnostics Port 2

Table B-8. Description of MS for Control Commands (Continued)

Terminal	Parameter Name	Recommended Value	Description
NO_BREC		0	
NO_INT		0	
NO_INTL		3	Number of Integer Long DAT references
NO_REAL		0	
USER		3	
SOURCE		SEND	
BLOCKED		1	Note: The MS must be blocked.
NET		7 - 11	CI535 submodule number. Record number for the CI535 data base element for the MVI Demo Protocol port defines the slave module number. Record no 1 => submodule = 7 Record no 2 => submodule = 8 Record no 3 => submodule = 9 (only valid for AC 450) Record no 4 => submodule = 10 (only valid for AC 450) Record no 5 => submodule = 11 (only valid for AC 450)
NODE		-3	Always -3 for Configuration MS
SCAN_FTR		1	Without function when BLOCKED=1
SORT_REF		YES	REF1 - 24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID		1	1 after first successful transmission. Reset by user.
ERROR			1 if the transmission has failed due to lost contact with destination node or queue full to CI535.
REF 1	Remote node	0	Not used by the MVI Demo Protocol Handler.
REF2	Auxiliary information 1	0	Not used by the MVI Demo Protocol Handler.
REF3	Auxiliary information 2	0	Not used by the MVI Demo Protocol Handler.

B.1.2.2 Read Commands

To update a value in Advant Controller 400, a Read Command must be issued to the corresponding slave unit. You can use a number of Read Command MS for this purpose.

The Read Command MS contain four DATs:

- The first DAT contains the start register address to read.
- The second DAT contains the number of registers (values) to read.
- The third DAT contains Auxiliary information 1.
- The fourth DAT contains Auxiliary information 2.

The data returned by the RTU is received by a receiving Data MS with the corresponding register address and IDENT=101-196 according to the Register Address MS with IDENT=4, 5, 6, 7 (port 1) or 14, 15, 16, 17 (port 2).

No Read Command MS are to be created when CI535 operates as a slave on the MVI Demo Protocol link, since all data transfer is handled with commands from the master only.

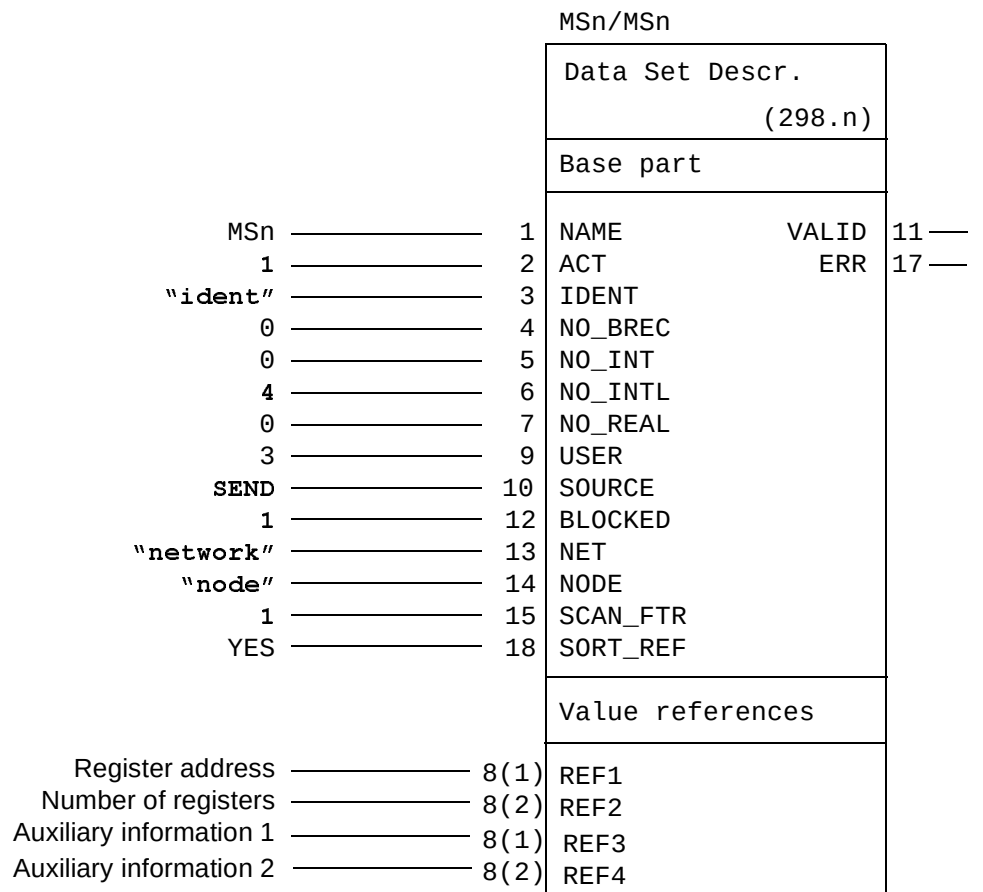


Figure B-10. Read Command MS

The parameters REF1 - REF4 are stored in DAT elements referred to by this MVI Data Set. You can give the DAT elements arbitrary names.

The parameters have the following function:

Table B-9. Description of MS for Read Command

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT			The identity of the MVI Data Set identifies the command code: 202 = Read Multiple Int16
NO_BREC		0	
NO_INT		0	
NO_INTL		4	Number of Integer Long DAT references
NO_REAL		0	
USER		3	
SOURCE		SEND	
BLOCKED		1	Note: The MS must be blocked.
NET		1 - 9	Network number of the MVI Demo Protocol bus. Defined in the CI535 data base element for the CI535 submodule.
NODE		1 - 99	RTU number (node) of the receiving RTU. Valid node numbers must be defined in the Network Configuration MS. Note: All the defined node numbers must be unique within the connected controller.
SCAN_FTR		1	Without function when BLOCKED=1.
SORT_REF		YES	REF1 - 24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID		1	1 after first successful transmission. Reset by user.
ERR			1 if the transmission has failed due to lost contact with destination node or queue full to CI535.
REF 1	Register address		The first register address to be read. This register address must be found in a MS defined in one of the Register Address MS, with IDENT = 4, 5, 6, 7 (port 1) or 14, 15, 16, 17 (port 2), in the master or the slave. The address is written by the PC program before the MS is sent.
REF2	Number of registers		Number of registers to be read. The PC program enters the number of registers before the Command MS is sent.
REF3	Auxiliary information 1	0	Protocol dependent information. This information is not used by the MVI Demo Protocol.
REF4	Auxiliary information 2	0	Protocol dependent information. This information is not used by the MVI Demo Protocol.

B.1.2.3 Write Commands

To update a value in a slave, a Write Command must be issued to the corresponding slave unit. A number of Command MS are used for this purpose.

The Write Command MS contain four DATs:

- The first DAT contains the start register address to write.
- The second DAT contains the number of registers (values) to write.
- The third DAT contains Auxiliary information 1.
- The fourth DAT contains Auxiliary information 2.

The data to be sent is fetched by the MVI Demo Protocol from a Data MS (IDENT=1-96) containing the defined registers according to the definition entered into the Register Address MS with IDENT = 4, 5, 6, 7 (port 1) or 14, 15, 16, 17 (port 2).

No Write Command MS are to be created when the MVI Demo Protocol operates as a slave on the MVI Demo Protocol link, since all data transfer is handled with commands from the master only.

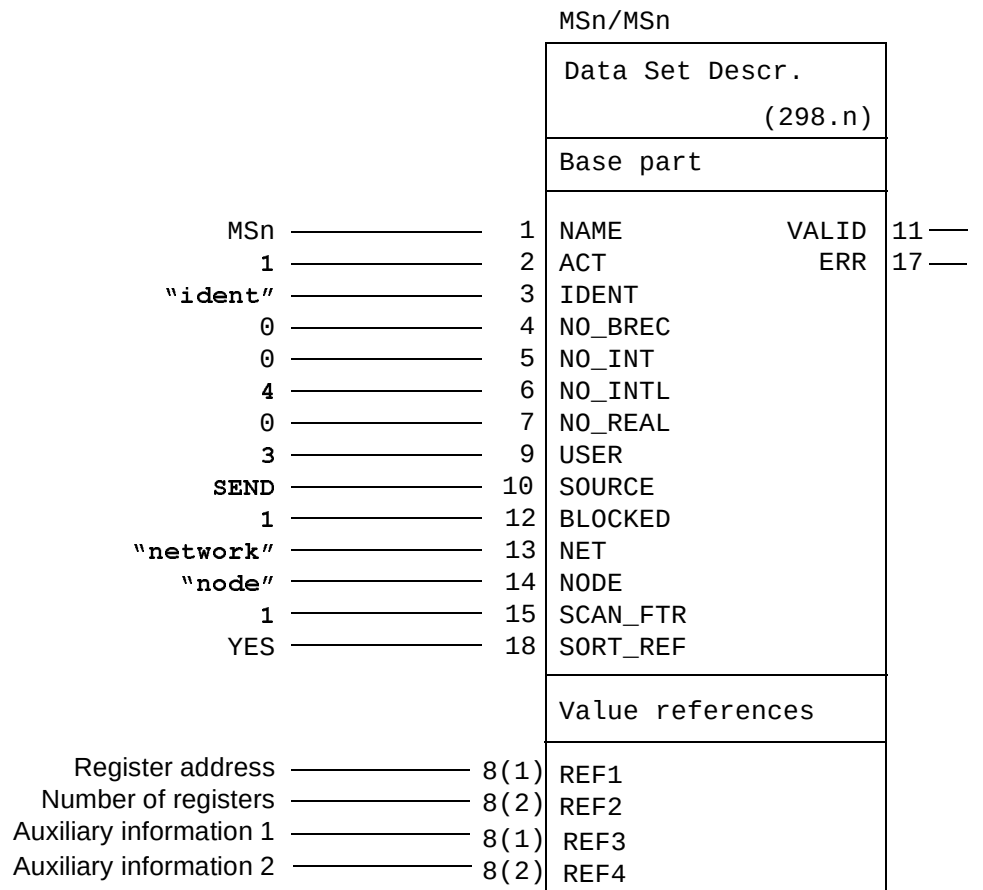


Figure B-11. Write Command MS

The parameters REF1 - REF4 are stored in DAT elements referred to by this MVI Data Set. You can give the DAT elements arbitrary names.

The parameters have the following function:

Table B-10. Description of MS for Write Command

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT			The identity of the MVI Data Set identifies the command code: 222 = Write Multiple Int16
NO_BREC		0	
NO_INT		0	
NO_INTL		4	Number of Integer Long DAT references
NO_REAL		0	
USER		3	
SOURCE		SEND	
BLOCKED		1	Note: The MS must be blocked.
NET		1 - 9	Network number of the MVI Demo Protocol bus. Defined in the CI535 data base element for the CI535 submodule.
NODE		1 - 99	RTU number (node) of the receiving RTU. Valid node numbers must be defined in the Network Configuration MS. Note: All the defined node numbers must be unique within the connected controller.
SCAN_FTR		1	Without function when BLOCKED=1.
SORT_REF		YES	REF1- 24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID		1	1 after first successful transmission. Reset by user.
ERR			1 if the transmission has failed due to lost contact with destination node or queue full to CI535.
REF 1	Register address		The first register address to be written. This register address must be found in a MS defined in one of the Register Address MS, with IDENT = 4, 5, 6, 7 (port 1) or 14, 15, 16, 17 (port 2), in the master or the slave. The address is written by the PC program before the MS is sent.
REF2	Number of registers		The number of registers to be written. The PC program enters the number of registers before the Command MS is sent.
REF3	Auxiliary information 1	0	Protocol dependent information. This information is not used by the MVI Demo Protocol.
REF4	Auxiliary information 2	0	Protocol dependent information. This information is not used by the MVI Demo Protocol.

B.1.2.4 Data MS for Data Transfer

Data Transmission to an RTU

MS for transmission of data to an RTU are defined in the same way as normal sending Data Set. The difference is that the MS used for data transmission on MVI Demo Protocol must be blocked (BLOCKED = 1) and USER = 3.

Master Mode

The transmission of the MS is controlled by the Write Command MS (IDENT = 222) together with the CI535 submodule.

The transmission is controlled by a PC program which sends a Write Command MS to the CI535 submodule. The CI535 submodule requests the sending Data MS containing the addressed registers and transmits them to the addressed RTU.

Slave Mode

The transmission of the Data MS is controlled by a Read Command from the MVI Demo Protocol master function code 1-4 together with the CI535 submodule.

The CI535 submodule requests the sending Data MS containing the addressed registers and returns the defined registers to the master.

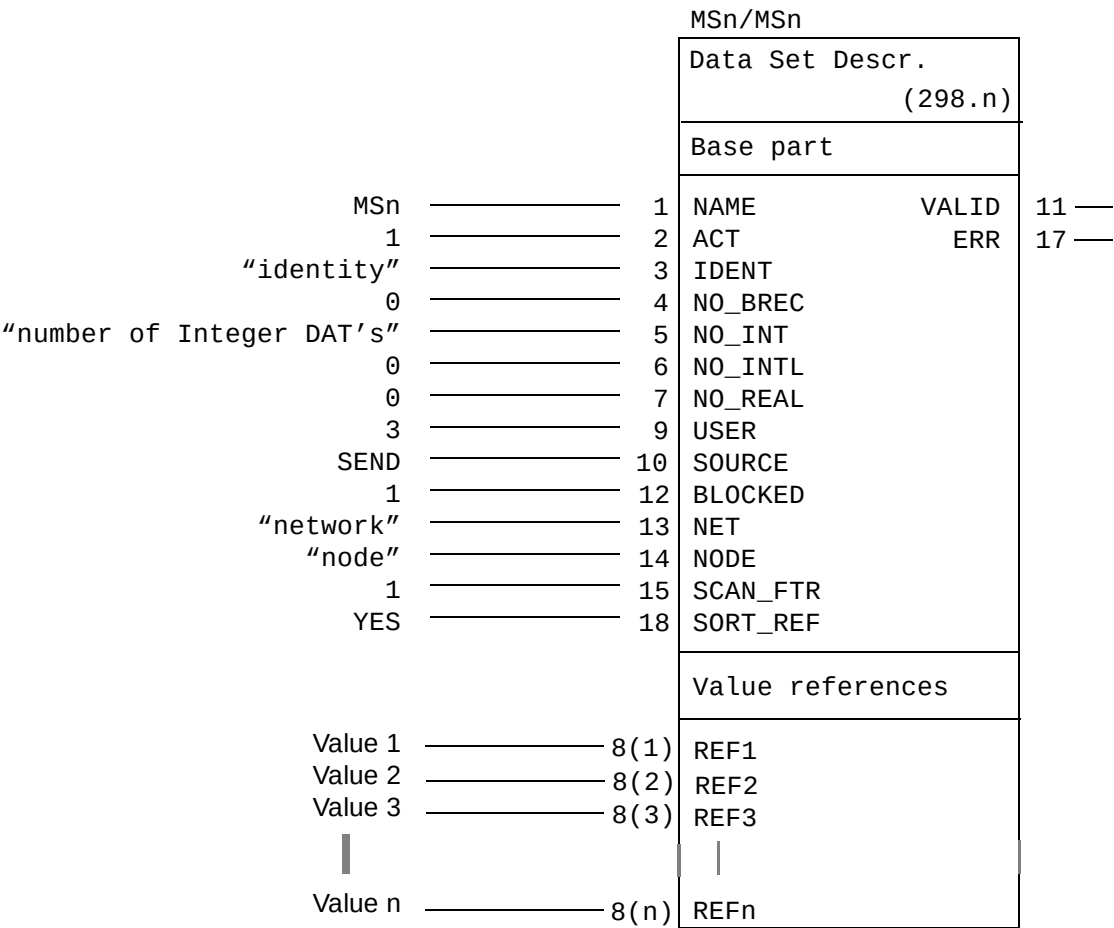


Figure B-12. MVI Data Set for Transmission of Data

The parameters REF1 - REF24 are stored in DAT elements referred to by this MVI Data Set. You can give the DAT elements arbitrary names.

The parameters have the following function:

Table B-11. Description of MS for Transmission of Data

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT			The identity of the MVI Data Set (1-96)
NO_BREC		0	Not used.
NO_INT		0-24	Number of Integer DAT references.
NO_INTL		0	Not used.
NO_REAL		0	Not used.
USER		3	
SOURCE		SEND	
BLOCKED		1	Note: The MS must be blocked.
NET		1 - 9	Network number of the MVI Demo Protocol bus. Defined in the CI535 data base element for the CI535 submodule.
NODE		1 - 99	RTU number (node) of the receiving RTU. Valid node numbers must be defined in the Network Configuration MS. Note: All the defined node numbers must be unique within the connected controller.
SCAN_FTR		1	Without function when BLOCKED=1.
SORT_REF		YES	REF1 - 24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID		1	1 after first successful transmission. Reset by user.
ERR			1 if the transmission has failed due to lost contact with destination node or queue full to CI535.
REF 1	Value 1		DAT element to be sent to the RTU.
REF2	Value 2		DAT element to be sent to the RTU.
REF3	Value 3		DAT element to be sent to the RTU.
REFn	Value n		DAT element to be sent to the RTU.

Data Transmission from an RTU

MS for reception of data from RTUs are defined in the same way as normal receiving Data Sets (DS), except USER=3.

NOTE

Every receiving Data MS must have a corresponding sending Data MS, to make the DATs accessible for the communication handler on the CI535 submodule. These MS must have IDENT = (IDENT (of receiving MS) - 100). In order to update only a part of a Data MS, there must always be a sending Data MS (IDENT = 1-96) corresponding to a receiving Data MS (IDENT = 101-196). The sending and receiving Data MS are connected to the same DAT elements. When the CI535 submodule receives data from the RTU, the sending Data MS is requested from Advant Controller 400. The received data from the RTU is mapped into the data from the sending Data MS before it is sent back to the receiving Data MS in the Advant Controller 400.

Master Mode

The data is requested from the RTU with a Read Command MS (IDENT = 202) together with the CI535 submodule.

The transmission is controlled by a PC program which sends a Read Command MS to the CI535 submodule. The CI535 submodule requests the sending Data MS containing the addressed registers and transmits the corresponding Read Command to the addressed RTU.

The reply from the RTU is mapped into the previously requested Data MS and returned to the corresponding receiving Data MS.

Slave Mode

A Write Command (function code 2) is received from the master containing data. The CI535 submodule requests the sending MS containing the addressed registers. The data from the master is mapped into the previous requested MS and returned to the corresponding receiving MS.

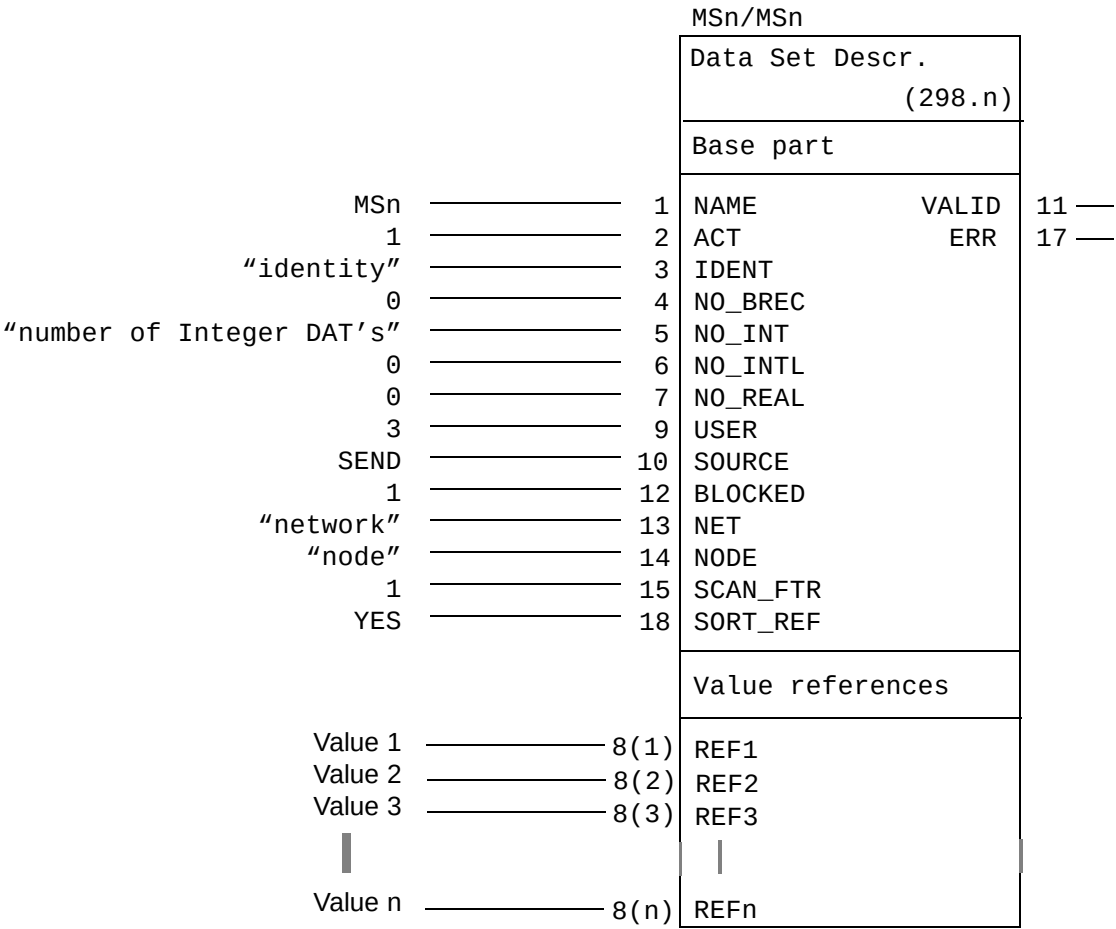


Figure B-13. MVI Data Set for Reception of Data

The parameters REF1 - REF24 are stored in DAT elements referred to by this MVI Data Set. You can give the DAT elements arbitrary names.

The parameters have the following function:

Table B-12. Description of MS for Reception of Data

Terminal	Parameter Name	Recommended Value	Description
NAME			Arbitrary Name of the MVI Data Set
ACT		1	1 = Active, 0 = Inactive
IDENT			The identity of the MVI Data Set (101-196)
NO_BREC		0	Not used.
NO_INT		0 - 24	Number of Integer DAT references.
NO_INTL		0	Not used.
NO_REAL		0	Not used.
USER		3	
SOURCE		RECEIVE	
BLOCKED		0	
NET		1 - 9	Network number of the MVI Demo Protocol bus. Defined in the CI535 data base element for the CI535 submodule.
NODE		1 - 99	RTU number (node) of the receiving RTU. Valid node numbers must be defined in the Network Configuration MS. Note: All the defined node numbers must be unique within the connected controller.
SCAN_FTR		1	Used for reset of the VALID flag. VALID is set to 0 when the Time = 3*SCAN_FTR * MS_SCANTIME has passed. Default value for MS_SCANTIME = 2 s.
SORT_REF		YES	REF1 - 24 sorted with Boolean DATs first, then Integer and long Integer and, finally, Real DATs.
VALID		1	1 when the MS is updated. See also SCAN_FTR above.
ERROR			1 if wrong data size is received.
REF 1	Value1		DAT element from the RTU.
REF2	Value 2		DAT element from the RTU.
REFn	Value n		DAT element from the RTU.

Transfer of Appended Data MS

The MVI Demo Protocol functions (Command MS with IDENT=201-210 and 221-230) will support data transfer from more than one Data MS in the same MVI Demo Protocol message, i.e., handling of more than 24/768 registers. If these MVI Demo Protocol functions must request or transmit data from two or more Data MS, the following must be accomplished:

- The MVI Demo Protocol register addresses must be consecutive in all Data MS used for one command (no holes are permitted). The last MVI Demo Protocol register address for the first Data MS must be followed directly by the start register address for the second Data MS, the last MVI Demo Protocol register address for the second Data MS must be followed directly by the start register address for the third Data MS, etc.
- All Data MS, except the last Data MS, used for one command **must** be built with 24 DATs.

PC Program Layout

To control the data flow on the asynchronous communication link, you must build a PC program in the Adant Controller 400. The PC program receives status information from the MVI Demo Protocol.

The flow control is maintained through a chain of SENDREQ PC elements, each corresponding to a Command MS to be transmitted. See [Section 3.6.7.4, Type Circuits for MVI Protocol Data Flow Control](#).

Appendix C A Makefile Example

C.1 Makefile

```
#-----  
#  
# FILE          Makefile  
#  
#  
# DESCRIPTION    Makefile for the MVI Modbus Protocol Handler subsystem.  
#  
# AUTHOR        SEISY/LKL Lars Larsson  
#  
# All rights reserved. Reproduction, modification, use or disclosure  
# to third parties without express authority is forbidden.  
# Copyright ABB Industrial Systems AB, Sweden, 1997.  
#  
# HISTORY:  
#  
#  
# ABSTRACT  
#   This file is used by Gnumake to build the MVI Demo Protocol Handler.  
#  
#-----  
  
COMPFLAGS      =  
  
CFLAGS         = -c -g -m68000 -ansi -pedantic -Wall -D CPU=MC68000 \  
                -D VxWorks -fvolatile -I/ipa/products/MVIBase/1.0-0/include  
  
PFLAGS         = $(COMPFLAGS)  
FFLAGS         = $(COMPFLAGS)  
CXXFLAGS       = $(COMPFLAGS)  
  
DEST           = .  
  
EXTHDRS        =  
  
HDRS           = mviDemo.h  
GCC            = cc68k  
LD             =  
LDFLAGS        = $(COMPFLAGS)  
LIBS           =  
LINTLIBS       =  
LINTFLAGS      = -u $(CFLAGS)  
MAKEFILE       = Makefile
```

```
OBJS                = mviDemo.o
PRINT               = pr
PRINTFLAGS          =
LP                  = lp
LPFLAG              =
PROGRAM             = dermoHandler
SHELL               = /bin/sh
SRCS                = mviDemo.c
all:                $(OBJS)
$(OBJS):             $(HDRS) $(EXTHDRS) $(SRCS) $(MAKEFILE)
                    $(GCC) $(CFLAGS) $(SRCS)

clean;;             @rm -f $(OBJS) core
clobber;;           @rm -f $(OBJS) $(PROGRAM) core tags
depend;;            @mkmf -f $(MAKEFILE) ROOT=$(ROOT)
echo;;              @echo $(HDRS) $(SRCS)
index;;             @ctags -wx $(HDRS) $(SRCS)
install:            $(PROGRAM)
                    @echo Installing $(PROGRAM) in $(DEST)
                    @-strip $(PROGRAM)
                    @if [ $(DEST) != . ]; then \
                    (rm -f $(DEST)/$(PROGRAM); $(INSTALL) -f $(DEST) $(PROGRAM));
                    fi

lint:               $(LINTLIBS) $(SRCS) $(HDRS) $(EXTHDRS)
                    lint $(LINTFLAGS) $(LINTLIBS) $(SRCS)
print;;             @$(PRINT) $(PRINTFLAGS) $(HDRS) $(SRCS) | $(LP) $(LPFLAGS)
tags:               $(HDRS) $(SRCS); @ctags $(HDRS) $(SRCS)
update:            $(SRCS) $(LIBS) $(HDRS) $(EXTHDRS)
                    @$(MAKE) -f $(MAKEFILE) ROOT=$(ROOT) DEST=$(DEST) install
touch;;             @touch $(SRCS) $(HDRS)

###
mviDemo.o: mviDemo.h
```

Appendix D Hardware Module

D.1 CI535 Communication Interface

CI535 communication interface has:

- Two RS-232-C communication interfaces
- Modem support.

Description

CI535 Multi Vendor Interface is a submodule which is to be mounted in the submodule carriers SC510 and SC520 in Advant Controller 450 and Processor Module PM150 in Advant Controller 410. The two RS-232-C communication interfaces are used for communication with asynchronous communication protocols.

The communication channels support communication speeds up to 19.2 kbit/s, which is the limit set by the system software. Both channels can run at this speed simultaneously.

The maximum communication distance without modems is 15 m. The modem signals which are supported can be found in [Table D-1](#) below.

Communication PINs are short-circuit proof.

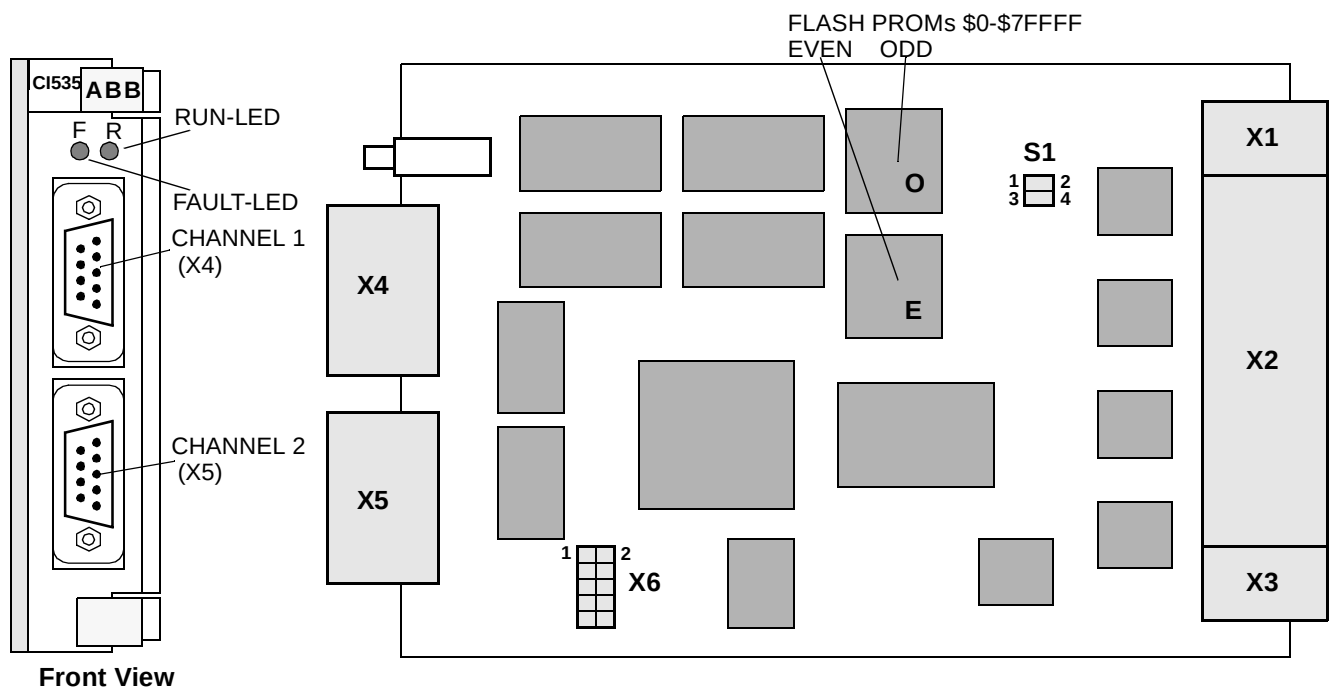


Figure D-1. CI535 Submodule

Technical Data

Indicators

LED R, Run (green) on module front. Indicating module running normally.

LED F, Fault (red) on module front. Indicating a fatal error detected on the CI535 module.
The LED is also turned on at reset of the module.

Jumpers

The board contains one jumper for special purposes. In normal operation always keep the S1 in position 3 - 4 (“parking place”). The component and position indications are found on the printed circuit board.

Connectors

Serial channels 1 and 2 connectors (X4 and X5):

- Connector type 9-pole male DSUB (DE9P)
- Placement On module front
- Pin designation See [Table D-1](#) below.

Table D-1. Pin Designation for Channels 1 and 2, Connectors X4 and X5

Pin	Short	Description
1	DCD	Data Carrier Detect
2	RD	Receive Data
3	TD	Transmit Data
4	DTR	Data Terminal Ready
5	GND	Ground
6	DSR	Data Set Ready
7	RTS	Request To Send
8	CTS	Clear To Send
9	RI	Ring Indicator

Power Supply

5V	typical	400 mA
	max.	670 mA
24V	max.	40 mA
Power loss (heat)	typical	3 W

Mechanical Data

Module size: Occupying one submodule slot (H = 95 mm, L = 140 mm, connectors not included).

Weight: 0.13 kg

D.2 CI536 Communication Interface

CI536 communication interface has:

- Two RS-232-C communication interfaces
- Modem support.

Description

CI536 Multi Vendor Interface is a submodule which is to be mounted in the submodule carriers SC510 and SC520 in Advant Controller 450 and Processor Module PM150 in Advant Controller 410. The two RS-232-C communication interfaces are used for communication with asynchronous communication protocols.

The communication channels support communication speeds up to 19.2 kbit/s, which is the limit set by the system software. Both channels can run at this speed simultaneously.

The maximum communication distance without modems is 15 m. The modem signals which are supported can be found in [Table D-1](#) below.

Communication PINs are short-circuit proof.

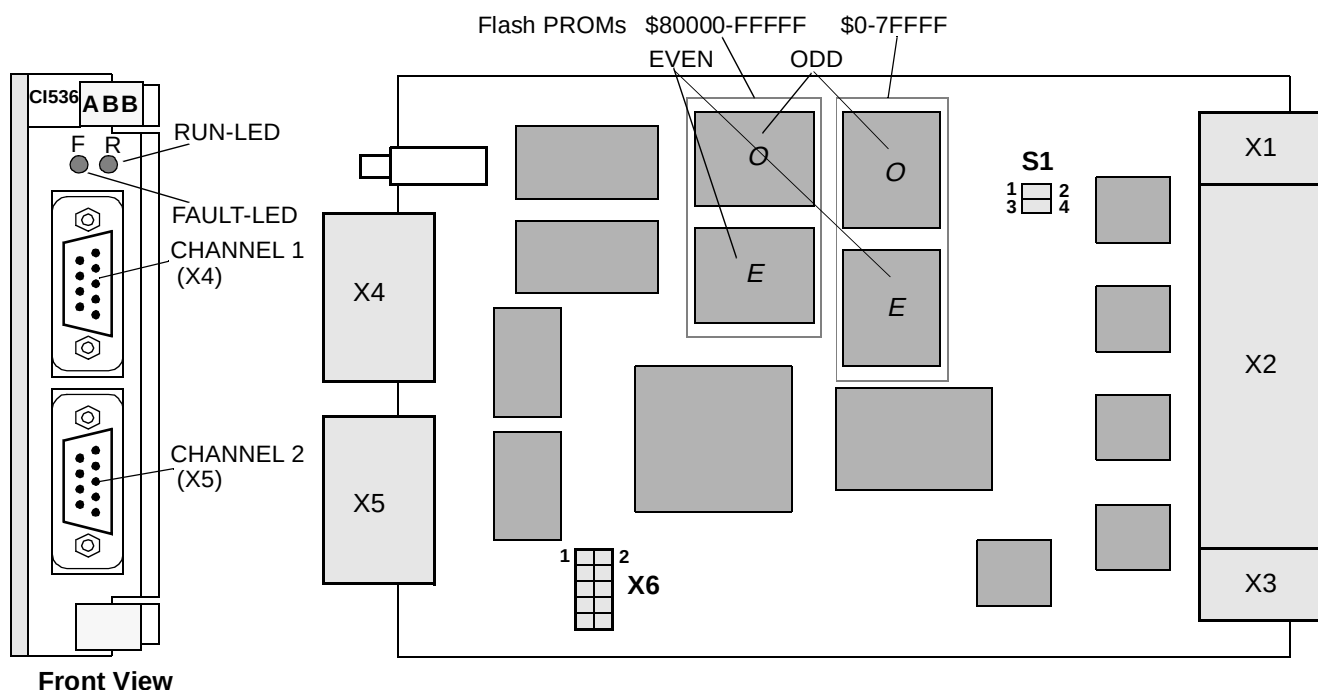


Figure D-2. CI536 Submodule

Technical Data

Indicators

LED R, Run (green) on module front. Indicating module running normally.

LED F, Fault (red) on module front. Indicating a fatal error detected on the CI536 module.
The LED is also turned on at reset of the module.

Jumpers

The board contains one jumper for special purposes. In normal operation always keep the S1 in position 3 - 4 ("parking place"). The component and position indications are found on the printed circuit board.

Connectors

Serial channels 1 and 2 connectors (X4 and X5):

- Connector type 9-pole male DSUB (DE9P)
- Placement On module front
- Pin designation See [Table D-1](#) below.

Table D-2. Pin Designation for Channels 1 and 2, Connectors X4 and X5

Pin	Short	Description
1	DCD	Data Carrier Detect
2	RD	Receive Data
3	TD	Transmit Data
4	DTR	Data Terminal Ready
5	GND	Ground
6	DSR	Data Set Ready
7	RTS	Request To Send
8	CTS	Clear To Send
9	RI	Ring Indicator

Power Supply

5V	typical	400 mA
	max.	670 mA
24V	max.	40 mA
Power loss (heat)	typical	3 W

Mechanical Data

Module size: Occupying one submodule slot (H = 95 mm, L = 140 mm, connectors not included).

Weight: 0.13 kg

INDEX

Symbols

/dev/ni 5-8
/dev/ttyxx 5-8

A

AC 400 1-7
Activation of commands and flow control 4-1
Advant Controller 410 1-2, 1-4
Advant Controller 450 1-2
AMPL 1-7
Application building
 Data flow control 3-133
 PC program sequence 3-133
 PC program type circuit 3-133
 SENDREQ 3-133
Area code 3-98, B-9

B

Bitrate 3-91, B-4
bufAllocate 3-137
bufRelease 3-138

C

Cables 1-3
 Cable for MVI communication 2-7
 Cable for SLIP communication 2-9
Capacity 3-2
CCITT V.24 2-7
Char time-out 3-91, B-4
Character length 3-91, B-4
checkStack 5-1
Checksum library
 mviAddControl 3-47
 mviBcc 3-43
 mviCrc_16 3-39
 mviCrc_8 3-45
 mviLrc 3-44
 mviSkipControl 3-46
 mviSum 3-45
CI535 2-3, 2-11
close 3-142
close, system call 3-29, 3-36
Command MS 1-7, 3-23

Commands

Control commands 3-117, B-20
Read commands 3-119, B-23
Write commands 3-122, B-25

Compiler 5-9

Configuration 3-88

Data flow control 3-133
Demo protocol 3-22
Development tools 3-3
Download script 3-8
Internet address 3-3
PC program sequence 3-133
PC program type circuit 3-133
PPL 3-5
SENDREQ 3-133
SLIP 3-5
Test directory 3-8

Configuration debugger 3-9

Configuration mode 4-1

Configuration MS 1-7, 3-23, 3-88, B-1

Control commands 3-117, B-20

Controller 1-7

D

DAT 1-7

Data base element

CI535 2-3

Data flow control 3-133

Data link handler 3-1

Data link interface 3-81

mviGetDLIDownCmnd 3-84
mviGetDLIDownResp 3-85
mviGetDLIUpCmnd 3-86
mviGetDLIUpResp 3-87
mviSendDLIDownCmnd 3-83
mviSendDLIDownResp 3-84
mviSendDLIUpCmnd 3-85
mviSendDLIUpResp 3-86

Data MS 1-7, 3-23

Data transfer B-32

Data transmission from an RTU 3-128, B-29

Data transmission to an RTU 3-125, B-27

Debugger 3-9, 5-9

Demo protocol 1-10, 3-22, 5-10

Bitrate B-4

Char time-out B-4

Character length B-4

Character time-out B-4

Command MS 3-23

Configuration MS 3-23, B-1

Data MS 3-23

Data transmission from an RTU B-29

Data transmission to an RTU B-27

Duplex B-4

Full duplex B-4

Function codes 3-24

Half duplex B-4

Line characteristics B-1

Line characteristics MS 3-23

Master mode B-27, B-29

NET B-3

Network configuration B-6

Network configuration MS 3-23

NODE B-3

Parity B-4

PC program layout B-32

Poll cycle time B-5

Post-idle time B-4

Pre-idle time B-4

Read registers 3-24

Register address MS 3-23

Register addresses MS B-17

Retransmissions B-5

Slave B-4

Slave mode B-27, B-29

Status MS 3-23

Status1 B-13

Status2 B-14

Status3 B-14

Stopbits B-4

Transfer of appended Data MS B-32

Turn-around time B-5

Write registers 3-24

Development tools 3-3

Direct connection 3-6

Download script 3-8

DS 1-7

Duplex 3-91, B-4

E

Editor 5-9

Engineering tools 1-4

F

Fault finding

/dev/ni 5-8

/dev/ttyxx 5-8

Compiler 5-9

Debugger 5-9

Demo protocol 5-10

Editor 5-9

Network interface device file 5-8

Remote services 5-8

rlogin 5-8

Serial port device file 5-8

SLIP 5-8

Target name 5-8

Test tool 5-11

VxGDB 5-9

Fault finding and user repair 5-8

Flow control 3-133

Full duplex 3-91, B-4

Function codes 3-24

Read registers 3-24

Write registers 3-24

Function layer 3-1

H

Half duplex 3-91, B-4

Hardware

Advant Controller 410 1-2, 1-4

Advant Controller 450 1-2

Cable for MVI communication 2-7

Cable for SLIP communication 2-9

Cables 1-3

Engineering tools 1-4

LED 5-1

Modems 1-3

Work station 1-3

Host interface 3-1

Host system interface 3-136
 bufAllocate 3-137
 bufRelease 3-138
 close 3-142
 Local address 3-137
 Network address 3-136
 O_RDONLY 3-140
 O_RDWR 3-140
 O_WRONLY 3-140
 open 3-140
 read 3-141
 Remote address 3-136
 tranChanDevCreate 3-139
 tranHostInfoGet 3-139
 Transceiver channel device 3-136
 transceiverInit 3-138
 write 3-141

I

Indicators
 LED indicators on CI535 5-1
Installation
 MVIBase 2-9
 PPL 2-11
 SDE 2-11
 SLIP 2-11
 User development structure 2-9
Internet address 3-3, 3-21
ioctl, system call 3-30, 3-36

L

LED 5-1
Line characteristics 3-88, B-1
Line characteristics MS 3-23, 3-88, B-1
Load module 3-21
Local address 3-137

M

Maintenance
 checkStack 5-1
 Stack usage 5-1
Makefile 3-18
Master 3-91, B-4
Master mode 3-125, 3-128, B-27, B-29
MasterBus 1-2, 1-7
Modems 1-3
MP 1-7
MS 1-7
MVI 1-1, 1-7

MVI development environment 1-1, 1-8
MVI protocol handler interface 3-17
MVI runtime environment 1-1, 1-9
MVI UART interface 3-25
MVI watchdog timer 3-35
mviAddControl 3-47
MVIBase 2-9, 3-15
mviBcc 3-43
MviCAssertDTR 3-32
MviCDeassertDTR 3-32
MviCGetBaudrate 3-32
MviCGetBreakHandling 3-32
MviCGetCharTimeout 3-32
MviCGetDataBits 3-32
MviCGetDuplex 3-31
MviCGetHalfDuplTmo 3-32
MviCGetParameters 3-30
MviCGetParity 3-32
MviCGetPostambleTime 3-32
MviCGetPreambleTime 3-32
MviCGetRxWakeupReason 3-32
MviCGetStopBits 3-32
MviCGetTxWakeupReason 3-32
MviCGetXonXoffHandl 3-32
MviCInitUART 3-32
mviCrc_16 3-39
mviCrc_8 3-45
mviCreateDLIDevices 3-56
mviCreatePHIDevices 3-55
mviCreateTimerDevices 3-57
mviCreateUARTDevice 3-51
MviCSetBaudrate 3-31
MviCSetBreakHandling 3-31
MviCSetCharTimeout 3-31
MviCSetDataBits 3-31
MviCSetDuplex 3-31
MviCSetHalfDuplTmo 3-31
MviCSetParameters 3-30
MviCSetParity 3-31
MviCSetPostambleTime 3-31
MviCSetPreambleTime 3-31
MviCSetStopBits 3-31
MviCSetXonXoffHandl 3-31
MviCWDTCancel 3-36
MviCWDTStart 3-36
mviDLICleanup 3-61
mviDLIWaitForModemSignals 3-63
mviFetchSetupInfo 3-59
mviGetData 3-52

mviGetDLIDownCmnd 3-84
mviGetDLIDownResp 3-85
mviGetDLIUpCmnd 3-86
mviGetDLIUpResp 3-87
mviGetNodeDiagInfo 3-49
mviGetNodeDLEntry 3-48
mviGetNodeStatusInfo 3-49
mviGetPHIDiagInfo 3-74
mviGetPHIDownCmnd 3-78
mviGetPHIDownResp 3-79
mviGetPHILocalDownCmnd 3-76
mviGetPHILocalDownResp 3-77
mviGetPHIReqSetup 3-71
mviGetPHISetupInfo 3-72
mviGetPHIStatusInfo 3-75
mviGetPHIUpCmnd 3-80
mviGetPHIUpResp 3-81
mviGetPHIWatchdog 3-73
mviInitDLStruct 3-59
mviInitGlobalDataStruct 3-60
mviInitUART 3-50
mviLrc 3-44
mviMakeWDTDevName 3-60
mviOpenDLIDevices 3-57
mviOpenPHIDevices 3-56
mviOpenTimerDevices 3-58
mviOpenUARTDevice 3-51
mviPHIWaitForModemSignals 3-62
mviReqData 3-51
mviReqPHISetup 3-70
mviSelect 3-53
mviSendData 3-52
mviSendDLIDownCmnd 3-83
mviSendDLIDownResp 3-84
mviSendDLIUpCmnd 3-85
mviSendDLIUpResp 3-86
mviSendLinkStatus 3-55
mviSendPHIDiagInfo 3-73
mviSendPHIDownCmnd 3-77
mviSendPHIDownResp 3-78
mviSendPHILocalDownCmnd 3-75
mviSendPHILocalDownResp 3-76
mviSendPHISetupInfo 3-71
mviSendPHIStatusInfo 3-74
mviSendPHIUpCmnd 3-79
mviSendPHIUpResp 3-80
mviSendPHIWatchdog 3-72
mviSkipControl 3-46
mviSum 3-45

N

NET 2-5, 3-90, B-3
Network address 3-136
Network configuration 3-95, B-6
Network configuration MS 3-23
Network interface device file 5-8
Network layer 3-1
NODE 2-4, 3-90, B-3

O

O_NDELAY 3-28
O_RDONLY 3-28, 3-140
O_RDWR 3-28, 3-140
O_WRONLY 3-28, 3-140
open 3-140
open, system call 3-28, 3-35
Operation mode 4-1

P

Parity 3-91, B-4
PC program 1-7, B-32
PC program layout 3-131
PC program sequence 3-133
PC program type circuit 3-133
Performance 3-2
Poll cycle time 3-92, B-5
POSITION 2-4
Post-idle time 3-91, 3-95, B-4, B-6
PPL 2-11, 3-5
 Direct connection 3-6
Pre-idle time 3-91, 3-95, B-4, B-6
Program builder 3-18
projm 3-11
Protocol design
 Data link handler 3-1
 Function layer 3-1
 Host interface 3-1
 Network layer 3-1
 Start-up function 3-1
Protocol development
 Makefile 3-18
 MVI protocol handler interface 3-17
 MVIBase 3-15
 Program builder 3-18
 Protocol handler interface 3-17
 SDE 3-17
 SDE/SoftBench program builder 3-18
 VxGDB 3-19

Protocol handler interface 3-17, 3-65
 mviGetPHIDiagInfo 3-74
 mviGetPHIDownCmnd 3-78
 mviGetPHIDownResp 3-79
 mviGetPHILocalDownCmnd 3-76
 mviGetPHILocalDownResp 3-77
 mviGetPHIReqSetup 3-71
 mviGetPHISetupInfo 3-72
 mviGetPHIStatusInfo 3-75
 mviGetPHIUpCmnd 3-80
 mviGetPHIUpResp 3-81
 mviGetPHIWatchdog 3-73
 mviReqPHISetup 3-70
 mviSendPHIDiagInfo 3-73
 mviSendPHIDownCmnd 3-77
 mviSendPHIDownResp 3-78
 mviSendPHILocalDownCmnd 3-75
 mviSendPHILocalDownResp 3-76
 mviSendPHISetupInfo 3-71
 mviSendPHIStatusInfo 3-74
 mviSendPHIUpCmnd 3-79
 mviSendPHIUpResp 3-80
 mviSendPHIWatchdog 3-72

R

read 3-141
Read commands 3-119, B-23
Read registers 3-24
Record number 2-4
Register addresses MS 3-23, 3-107, B-17
 Data MS relationship 3-109
Remote address 3-136
Remote services 5-8
Requirements
 Advant Controller system 2-1
 Development system 2-1
Retransmissions 3-92, B-5
rlogin 5-8
RTU 1-7
RTU number 3-98, B-9
RTU status 3-99, B-9
 Status1 3-103, B-13
 Status2 3-103, B-14
 Status3 3-103, B-14

Runtime library 3-47
 mviCreateDLIDevices 3-56
 mviCreateTimerDevices 3-57
 mviCreateUARTDevice 3-51
 mviDLICleanup 3-61
 mviDLIWaitForModemSignals 3-63
 mviFetchSetupInfo 3-59
 mviGetData 3-52
 mviGetNodeDiagInfo 3-49
 mviGetNodeDLEntry 3-48
 mviGetNodeStatusInfo 3-49
 mviInitDLStruct 3-59
 mviInitGlobalDataStruct 3-60
 mviInitUART 3-50
 mviMakeWDTDevName 3-60
 mviOpenDLIDevices 3-57
 mviOpenPHIDevices 3-56
 mviOpenTimerDevices 3-58
 mviOpenUARTDevice 3-51
 mviPHIWaitForModemSignals 3-62
 mviReqData 3-51
 mviSelect 3-53
 mviSendData 3-52
 mviSendLinkStatus 3-55

Runtime operation

 Activation of commands and flow control 4-1
 Configuration mode 4-1
 Operation mode 4-1
 Status information 4-1

S

SDE 1-7, 2-11, 3-17
SDE/SoftBench program builder 3-18
select, system call 3-33, 3-37
SENDREQ 3-133
Serial port device file 5-8
SERVICE 2-11
Shut-down system
 CI535 2-11
 SERVICE 2-11
Slave 3-91, B-4
Slave Functions 3-24
Slave mode 3-126, 3-128, B-27, B-29
SLIP 2-11, 3-5, 5-8
 Direct connection 3-6
Software
 Work station 1-3
S-record file 3-21
Stack usage 5-1

- Start-up function 3-1
- Start-up system
 - CI535 2-11
 - SERVICE 2-11
- Status information 4-1
- Status MS 3-23
- Status1 3-103, B-13
- Status2 3-103, B-14
- Status3 3-103, B-14
- Stopbits 3-91, B-4
- struct timeval 3-33
- Submodule number 2-4
- SUBPOS 2-4
- Subscriber number 3-98, B-9
- System generation
 - Internet address 3-21
 - Load module 3-21
 - S-record file 3-21
- system 3-11

T

- Target name 5-8
- Test directory 3-8
- Test tool 5-11
- TK577 2-7
- TK595 2-7
- tranChanDevCreate 3-139
- tranHostInfoGet 3-139
- Transceiver channel device 3-136
- transceiverInit 3-138
- Transfer of appended Data MS 3-131, B-32
- Transmission speed 3-91, B-4
- Turn-around time 3-92, 3-95, B-5 to B-6
- Type circuit 3-133

U

- UART interface 3-25
 - close, system call 3-29
 - ioctl, system call 3-30
 - MviCAssertDTR 3-32
 - MviCDeassertDTR 3-32
 - MviCGetBaudrate 3-32
 - MviCGetBreakHandling 3-32
 - MviCGetCharTimeout 3-32
 - MviCGetDataBits 3-32
 - MviCGetDuplex 3-31
 - MviCGetHalfDuplTmo 3-32
 - MviCGetParameters 3-30
 - MviCGetParity 3-32

- MviCGetPostambleTime 3-32
- MviCGetPreambleTime 3-32
- MviCGetRxWakeupReason 3-32
- MviCGetStopBits 3-32
- MviCGetTxWakeupReason 3-32
- MviCGetXonXoffHandl 3-32
- MviCInitUART 3-32
- MviCSetBaudrate 3-31
- MviCSetBreakHandling 3-31
- MviCSetCharTimeout 3-31
- MviCSetDataBits 3-31
- MviCSetDuplex 3-31
- MviCSetHalfDuplTmo 3-31
- MviCSetParameters 3-30
- MviCSetParity 3-31
- MviCSetPostambleTime 3-31
- MviCSetPreambleTime 3-31
- MviCSetStopBits 3-31
- MviCSetXonXoffHandl 3-31
- O_NDELAY 3-28
- O_RDONLY 3-28
- O_RDWR 3-28
- O_WRONLY 3-28
- open, system call 3-28
- select, system call 3-33
- struct timeval 3-33

- User development structure 2-9

V

- VxGDB 3-9, 3-19, 5-9

W

- Watchdog timer 3-35
 - close, system call 3-36
 - ioctl, system call 3-36
 - MviCWDTCancel 3-36
 - MviCWDTStart 3-36
 - open, system call 3-35
 - select, system call 3-37
- Work station
 - Hardware 1-3
 - Software 1-3
- write 3-141
- Write commands 3-122, B-25
- Write registers 3-24



3BSE 000 531R0101 Rev A

February 1998